Università degli Studi di Firenze

Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione

Tesi di Laurea Triennale in Ingegneria Informatica

Tecniche di controllo per platooning di veicoli su traiettorie generiche

Candidato
Leonardo Del Bene

Relatore
Giorgio Battistelli

Correlatore

Anno Accademico 2023/2024

Indice

Elenco delle figure

Introduzione

Negli ultimi anni, l'evoluzione delle tecnologie di guida automatizzata ha portato a sviluppi significativi nel campo dei trasporti, con l'obiettivo di migliorare la sicurezza, l'efficienza energetica e la fluidità del traffico. Tra queste innovazioni, il platooning rappresenta un concetto avanzato e promettente. Il termine **"platooning"** si riferisce a una formazione di veicoli automatizzati che viaggiano in modo coordinato e ravvicinato, mantenendo una distanza minima tra di loro grazie a sistemi di comunicazione e controllo avanzati.

L'aumento delle richieste di trasporto, non compensato da una crescita adeguata della capacità autostradale, porta inevitabilmente a un aumento del traffico congestionato. Per affrontare questa sfida, il *vehicle platooning* è stato sviluppato come una soluzione efficace per migliorare la capacità delle autostrade riducendo la distanza tra i veicoli. Il concetto di mantenere una distanza ottimale tra i veicoli è stato introdotto con l'invenzione del Controllo Adattivo della Velocità (ACC). L'ACC utilizza tecnologie come radar e lidar per misurare la distanza e la velocità relativa del veicolo che precede, regolando di conseguenza la velocità del veicolo seguente per mantenere la distanza desiderata. Come evoluzione dell'ACC, è stato sviluppato il Cooperative ACC (CACC), che integra la comunicazione veicolo-veicolo (V2V). Grazie alla comunicazione V2V, il veicolo seguente può ricevere informazioni

aggiuntive sul veicolo che lo precede, riducendo così la distanza tra i veicoli e attenuando le perturbazioni che si propagano all'indietro nel convoglio. Inoltre, mantenendo una distanza ravvicinata tra i veicoli, in particolare tra veicoli pesanti, è possibile ridurre la resistenza aerodinamica, con un conseguente abbattimento delle emissioni e del consumo di carburante per tutti i veicoli del convoglio.
Il *vehicle platooning*, che può essere descritto come una strategia di "seguire il leader", viene realizzato scambiando informazioni sulle proprietà longitudinali e laterali tra i veicoli.
Ogni veicolo nel convoglio è equipaggiato con un controllore, che ha il compito di generare la traiettoria del veicolo e di gestire l'accelerazione e la velocità angolare. Ogni veicolo deve seguire il predecessore aggiustando la propria velocità e orientamento.

La mia ricerca si focalizza sull'implementazione di leggi di controllo per la gestione della traiettoria dei veicoli e il mantenimento di una distanza di sicurezza dal veicolo che li precede all'interno di un convoglio. In particolare, l'attenzione è rivolta ai convogli di veicoli che percorrono tratti stradali curvilinei.
Nel progetto di tesi è stata implementate una leggi di controllo: il *look-ahead-based controller*; la quale presenta un problema noto come "taglio di curva": il controllo look-ahead può indurre il veicolo che segue a svoltare prematuramente quando rileva una discrepanza tra l'orientamento del veicolo che precede e quello che lo segue. Questo comportamento può portare il veicolo che segue a tagliare la curva, compromettendo la precisione della traiettoria.

Nel capitolo 1 sarà esaminata teoricamente la legge di controllo adottata.

Il capitolo 2 illustrerà l'implementazione della legge di controllo e lo sviluppo del software utilizzato per creare e simulare l'ambiente di un convoglio di veicoli su percorsi curvilinei. Successivamente, nel capitolo 3, verranno discussi i risultati ottenuti. Infine, le conclusioni saranno presentate nel capitolo 4.

Capitolo 1

Leggi di controllo

In questo capitolo verrà analizzato il processo di derivazione e implementazione della legge di controllo per la gestione dei veicoli in un sistema di platooning. Verranno descritti in dettaglio i modelli matematici utilizzati per rappresentare la dinamica dei veicoli e illustrata la derivazione analitica delle leggi di controllo.
La legge di controllo implementata in questo progetto di tesi è state sviluppata sulla base delle metodologie presentate in [1].

1.1 Modello cinematico

Il modello cinematico è descritto dalle seguenti equazioni differenziali:

$$\dot{x}_i = v_i \cos\theta_i \tag{1a}$$

$$\dot{y}_i = v_i \sin\theta_i \tag{1b}$$

$$\dot{v}_i = a_i \tag{1c}$$

$$\dot{\theta}_i = \omega_i \tag{1d}$$

dove (x_i, y_i) sono le coordinate cartesiane del veicoli i, θ_i è la orientazione del veicolo rispetto all'asse x, v_i è la velocità longitudinale, a_i è l'accelerazione longitudinale mentre ω_i è la velocità angolare.
L'obiettivo principale per il veicolo i all'interno di un convoglio è mantenere una distanza desiderata $d_{r,i}$ dal veicolo che lo precede,$i-1$. La distanza tra i veicoli può essere determinata utilizzando una politica di spaziatura basata su un time-gap costante, che varia in base alla velocità. La politica di spaziatura con time-gap costante viene nel presente lavoro formulata come...

$$d_{r,i} = \begin{bmatrix} d_{rx,i} \\ d_{ry,i} \end{bmatrix} = (r_i + h_i v_i) \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \end{bmatrix} \tag{2}$$

dove $d_{r,i}$ è il vettore della distanza desiderata tra il veicoli i e il veicolo $i-1$, r_i è la distanza di arresto e h_i è il time-gap. Il time-gap h_i può essere considerato come il tempo necessario al veicolo i per raggiungere la posizione attuale del veicolo che lo precede quando si sposta a velocità costante v_i.

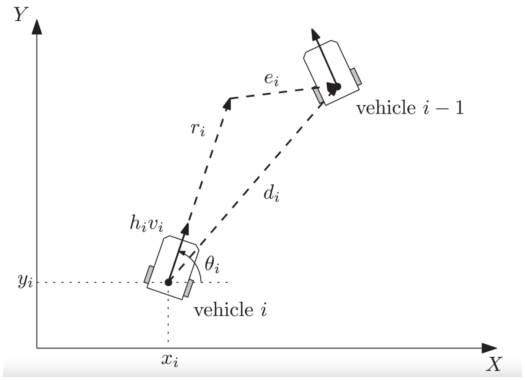

Figura 1.1: Definizione dell'errore in termini vettoriali per i veicoli che seguono, dove e_i indica l'errore di spaziatura e d_i rappresenta la distanza reale tra il veicolo i e il veicolo $i-1$

Poiché l'obiettivo principale del platooning dei veicoli è mantenere la distanza tra i veicoli, è logico derivare la dinamica dell'errore basandosi sulla differenza tra la distanza desiderata $d_{r,i}$ e la distanza effettiva tra i veicoli. L'errore di spaziatura viene definito come

$$e_i = d_i - d_{r,i} = (p_{i-1} - p_i) - d_{r,i} \tag{3}$$

con d_i la distanza attuale tra il veicolo i e il suo predecessore e $d_{r,i}$ come definito in (2).

1.2 Look-ahead-based controller

Considerando il modello appena descritto procediamo a definire le componenti dello stato come segue:

$$z_{1,i} = x_{i-1} - x_i - (r_i + h_i v_i)\cos\theta_i \tag{4a}$$

$$z_{2,i} = y_{i-1} - y_i - (r_i + h_i v_i)\sin\theta_i \tag{4b}$$

$$z_{3,i} = v_{i-1}\cos\theta_{i-1} - v_i\cos\theta_i \tag{4c}$$

$$z_{4,i} = v_{i-1}\sin\theta_{i-1} - v_i\sin\theta_i. \tag{4d}$$

Si osserva che la prima e la seconda equazione rappresentano l'errore e_i scomposto lungo le componenti x e y, rispettivamente. La terza e la quarta equazione, invece rappresentano l'errore della velocità lungo la direzione x e y.

Differenziando rispetto al tempo otteniamo:

$$\begin{bmatrix} \dot{z}_{1,i} \\ \dot{z}_{2,i} \end{bmatrix} = \begin{bmatrix} z_{3,i} \\ z_{4,i} \end{bmatrix} - F_i \begin{bmatrix} a_i \\ \omega_i \end{bmatrix} \tag{5a}$$

$$\begin{bmatrix} \dot{z}_{3,i} \\ \dot{z}_{4,i} \end{bmatrix} = H_{i-1} \begin{bmatrix} a_{i-1} \\ \omega_{i-1} \end{bmatrix} - H_i \begin{bmatrix} a_i \\ \omega_i \end{bmatrix} \tag{5b}$$

con:

$$F_i := \begin{bmatrix} h_i \cos\theta_i & -(r_i + h_i v_i)\sin\theta_i \\ h_i \sin\theta_i & (r_i + h_i v_i)\cos\theta_i \end{bmatrix} \tag{6a}$$

$$H_j := \begin{bmatrix} \cos\theta_j & -v_j \sin\theta_j \\ \sin\theta_j & v_j \cos\theta_j \end{bmatrix}, \quad j \in \{i-1, i\}. \tag{6b}$$

Il sistema (5) descrive la dinamica dell'errore interveicolare tra i veicoli i e $i-1$. L'obiettivo è ora progettare un ingresso di controllo $[a_i, \omega_i]^T$ che stabilizzi asintoticamente il sistema a zero.
Scegliendo:

$$\begin{aligned} \begin{bmatrix} a_i \\ \omega_i \end{bmatrix} &= F_i^{-1} \begin{bmatrix} z_{3,i} + k_{1,i} z_{1,i} \\ z_{4,i} + k_{2,i} z_{2,i} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{h_i}\cos\theta_i & \frac{1}{h_i}\sin\theta_i \\ -\frac{\sin\theta_i}{r_i + h_i v_i} & \frac{\cos\theta_i}{r_i + h_i v_i} \end{bmatrix} \begin{bmatrix} z_{3,i} + k_{1,i} z_{1,i} \\ z_{4,i} + k_{2,i} z_{2,i} \end{bmatrix} \end{aligned} \tag{7}$$

La dinamica dell'errore in anello chiuso si ottiene sostituendo (7) in (5):

$$\begin{bmatrix} \dot{z}_{1,i} \\ \dot{z}_{2,i} \end{bmatrix} = - \begin{bmatrix} k_{1,i} & 0 \\ 0 & k_{2,i} \end{bmatrix} \begin{bmatrix} z_{1,i} \\ z_{2,i} \end{bmatrix} \tag{8a}$$

$$\begin{bmatrix} \dot{z}_{3,i} \\ \dot{z}_{4,i} \end{bmatrix} = - G_i \begin{bmatrix} z_{3,i} \\ z_{4,i} \end{bmatrix} + \begin{bmatrix} \xi_{1,i} \\ \xi_{2,i} \end{bmatrix} \tag{8b}$$

con:

$$\begin{bmatrix} \xi_{1,i} \\ \xi_{2,i} \end{bmatrix} := H_{i-1} \begin{bmatrix} a_{i-1} \\ \omega_{i-1} \end{bmatrix} - G_i \begin{bmatrix} k_{1,i} z_{1,i} \\ k_{2,i} z_{2,i} \end{bmatrix} \tag{9}$$

$$G_i := \begin{bmatrix} \dfrac{h_i v_i + r_i \cos^2 \theta_i}{h_i (r_i + h_i v_i)} & \dfrac{r_i \sin \theta_i \cos \theta_i}{h_i (r_i + h_i v_i)} \\ \dfrac{r_i \sin \theta_i \cos \theta_i}{h_i (r_i + h_i v_i)} & \dfrac{h_i v_i + r_i \sin^2 \theta_i}{h_i (r_i + h_i v_i)} \end{bmatrix} \tag{10}$$

Da una prospettiva di platooning, uno svantaggio della strategia look-ahead è che la posizione laterale del veicolo che segue è accurata solo su percorsi rettilinei. Durante una manovra di svolta, questo porta a un problema noto come *"cutting-corner"*. Senza una conoscenza anticipata della traiettoria di riferimento, il controllo di posizione nella strategia look-ahead convenzionale può indurre i veicoli seguaci a curvare troppo presto. In queste situazioni, il veicolo i rileva un errore di posizione causato dalla politica di spaziatura e tende a curvare prima per correggerlo, piuttosto che aspettare di raggiungere il punto di svolta.

Capitolo 2

Implementazione delle leggi di controllo

In questo capitolo verrà presentata l'implementazione in Python della legge di controllo fondamentale per il sistema in esame. La scelta di Python come linguaggio di programmazione è stata motivata dalla sua versatilità, facilità d'uso e dalla vasta disponibilità di librerie specializzate per il calcolo numerico e la simulazione, quali NumPy, SciPy e Matplotlib.
Nelle sezioni seguenti, saranno descritte nel dettaglio le tecniche di implementazione adottate, l'obiettivo è fornire una chiara comprensione delle logiche sottostanti alla legge di controllo.
La legge di controllo illustrata e definita nel capitolo 1 è formulata con equazioni differenziali, pertanto opera in tempo continuo. Per renderla implementabile su un calcolatore, è stato necessario convertirla in equazioni alle differenze utilizzando il metodo di Eulero, adottando un passo di campionamento T.

2.1 Struttura del codice

2.1.1 Class diagram

Di seguito è illustrato il class diagram del codice sviluppato

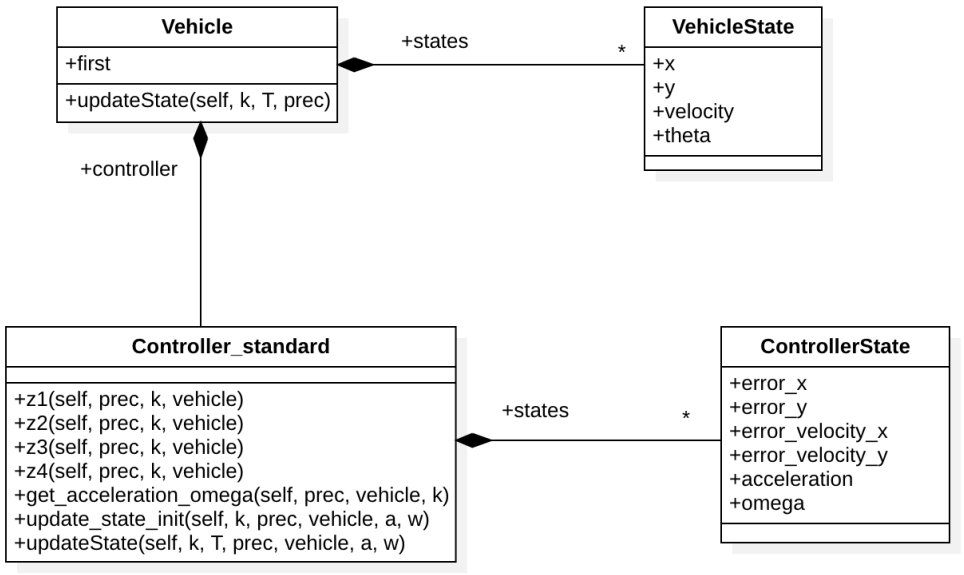

Figura 2.1: Class diagram

2.1.2 Struttura delle classi

Come illustrato nella figura 2.1, ogni veicolo del convoglio è dotato di una lista di stati, di un parametro booleano che indica se si tratta del primo veicolo del platooning e, infine, di un controllore.

I parametri del controllore includono due guadagni in ciclo chiuso, **k1** e **k2**, il parametro **r**, che indica la distanza di arresto, e il time-gap **h**, ovvero il tempo necessario al veicolo per raggiungere il predecessore mantenendo una velocità costante. Come per ogni veicolo, anche ogni controllore dispone di

una lista di stati. Dal class diagram in figura 2.1 possiamo osservare le classi *VehicleState* e *ControllerState*, che rappresentano rispettivamente gli stati dei veicoli e dei controllori.

Un singolo stato di un veicolo è definito dalla sua posizione lungo l'asse x, lungo l'asse y, dalla velocità e dall'orientazione del veicolo in un preciso istante. D'altro canto, un singolo stato del controllore è composto dall'errore tra la posizione del veicolo e quella del suo predecessore lungo l'asse x (*error_x*) e l'asse y (*error_y*), dall'errore di velocità sia lungo x sia lungo y, e infine dall'accelerazione e dalla velocità angolare (*omega*).

2.1.3 Aggiornamento dello stato dei veicoli e del controllore

Come già discusso, l'obiettivo del controllo è calcolare e applicare al veicolo i comandi di accelerazione e velocità angolare necessari per farlo seguire la traiettoria del veicolo precedente, mantenendo una distanza di sicurezza. Il controller produce questi input tramite la funzione *get_acceleration_omega(self, prec, vehicle, k)*. Poiché un calcolatore opera in tempo discreto, il tempo viene suddiviso in passi o iterazioni. Ad ogni nuovo passo, la prima operazione eseguita è l'aggiornamento della lista degli stati del controllore attraverso la funzione *updateState(self, k, T, prec, vehicle, a, w)*.

Questa operazione aggiunge un nuovo stato alla lista e aggiorna i relativi parametri, calcolando l'errore di posizione e di velocità. Tuttavia, questo processo non si applica al primo veicolo del convoglio, poiché essendo il leader, non ha un veicolo da seguire. Pertanto, per il leader, l'errore di posizione e di

velocità è pari a zero. Al leader viene assegnata una traiettoria predefinita, generata all'inizio dell'esecuzione del codice; di conseguenza, i parametri *a* (accelerazione) e *w* (velocità angolare) passati alla funzione *updateState(self, k, T, prec, vehicle, a, w)* rappresentano i valori che il controllore del leader deve generare per seguire la traiettoria desiderata. Questi segnali sono assegnati solo al leader, mentre per tutti gli altri veicoli, è il controllore del singolo veicolo a calcolarli.
Di seguito si trova il codice della funzione *updateState(self, k, T, prec, vehicle,a,w)* per il controller standard.

```
def updateState(self, k, T, prec, vehicle,a,w):
    self.states.append(ControllerState())
    if (prec != None):

        K1=np.array([
            [-self.k1 , 0],
            [0, -self.k2]
        ])

        Z1=np.array([
            [self.states[k-1].error_x],
            [self.states[k-1].error_y]
        ])

        Result1 = np.dot(K1,Z1)

        self.states[k].error_x = self.states[k - 1].
            error_x + (T * Result1[0, 0])
        self.states[k].error_y = self.states[k - 1].
```

```
            error_y + (T * Result1[1, 0])

        G = np.array([
            [(self.h * vehicle.states[k - 1].velocity +
                self.r * (math.cos(vehicle.states[k -
                1].theta) ** 2)) / (
                    self.h * (self.r + self.h * vehicle
                        .states[k - 1].velocity)),
                (self.r * math.sin(vehicle.states[k - 1].
                    theta) * math.cos(vehicle.states[k -
                    1].theta)) / (
                        self.h * (self.r + self.h *
                            vehicle.states[k - 1].velocity)
                            )],
            [(self.r * math.sin(vehicle.states[k - 1].
                theta) * math.cos(vehicle.states[k - 1].
                theta)) / (
                    self.h * (self.r + self.h * vehicle
                        .states[k - 1].velocity)),
                (self.h * vehicle.states[k - 1].velocity +
                    self.r * (
                        math.sin(vehicle.states[k - 1].
                            theta) ** 2)) / (
                        self.h * (self.r + self.h *
                            vehicle.states[k - 1].velocity)
                            )]
        ])

        Z = np.array([
```

```
        [self.states[k - 1].error_velocity_x],
        [self.states[k - 1].error_velocity_y]
    ])

    H = np.array([
        [math.cos(prec.states[k - 1].theta),
         -(prec.states[k - 1].velocity * math.sin(
             prec.states[k - 1].theta))],
        [math.sin(prec.states[k - 1].theta),
         (prec.states[k - 1].velocity * math.cos(
             prec.states[k - 1].theta))]
    ])

    A = np.array([
        [prec.controller.states[k - 1].acceleration
            ],
        [prec.controller.states[k - 1].omega]
    ])

    K = np.array([
        [self.k1 * self.states[k - 1].error_x],
        [self.k2 * self.states[k - 1].error_y]
    ])

    Result = (np.dot(-G, Z)) + ((np.dot(H, A) + np.
        dot(-G, K)))

    self.states[k].error_velocity_x = self.states[k
        - 1].error_velocity_x + (T * Result[0, 0])
```

```
            self.states[k].error_velocity_y = self.states[k
                - 1].error_velocity_y + (T * Result[1, 0])

        else:
            self.states[k].error_x = 0
            self.states[k].error_y = 0
            self.states[k].error_velocity_x = 0
            self.states[k].error_velocity_y = 0
            self.states[k].acceleration = a
            self.states[k].omega = w
```

Dopo l'aggiornamento dello stato del controllore, viene aggiornato anche lo stato del veicolo. Anche in questo caso, viene generato un nuovo stato da aggiungere alla lista degli stati del veicolo, e vengono aggiornati i parametri di posizione, velocità e orientazione che il veicolo deve assumere in quell'istante temporale. Questo aggiornamento viene eseguito utilizzando l'accelerazione e la velocità angolare calcolate dal controllore. Di seguito si può consultare il codice della funzione *updateState(self, k, T, prec)* della classe *Vehicle*

```
    def updateState(self, k, T, prec):
        self.states.append(VehicleState(0, 0, 0, 0))
        if (prec != None):
            A = self.controller.get_acceleration_omega(prec
                , self, k - 1)
            self.states[k].x = self.states[k - 1].x + (
                    T * self.states[k - 1].velocity * math.
                        cos(self.states[k - 1].theta))
            self.states[k].y = self.states[k - 1].y + (
                    T * self.states[k - 1].velocity * math.
                        sin(self.states[k - 1].theta))
```

```
            self.states[k].velocity = self.states[k - 1].
                velocity + (T * A[0, 0])
            self.states[k].theta = self.states[k - 1].theta
                + (T * A[1, 0])

        else:
            self.states[k].x = self.states[k - 1].x + (
                    T * self.states[k - 1].velocity * math.
                        cos(self.states[k - 1].theta))
            self.states[k].y = self.states[k - 1].y + (
                    T * self.states[k - 1].velocity * math.
                        sin(self.states[k - 1].theta))
            self.states[k].velocity = self.states[k - 1].
                velocity + (T * self.controller.states[k -
                1].acceleration)
            self.states[k].theta = self.states[k - 1].theta
                + (T * self.controller.states[k - 1].omega)
```

I metodi per aggiornare gli stati, sia del controller che del veicolo, vengono invocati nel seguente modo:

```
for k in range(1, num_steps):
    for i in range(N):
        vehicles[i].controller.updateState(k, T,
            vehicles[i - 1] if i > 0 else None, vehicles
            [i], acceleration[k],
                                               omega[k])
        vehicles[i].updateState(k, T, vehicles[i - 1]
            if i > 0 else None)
        state = vehicles[i].states[-1]
```

```
        x_positions[i].append(state.x)
        y_positions[i].append(state.y)
```

Dove N rappresenta il numero totale dei veicoli nel platooning e k l'istante temporale. Si noti che il ciclo più esterno inizia da k=1 e termina a num_steps, poiché l'inizializzazione per k=0 viene eseguita separatamente."

2.1.4 Inizializzazione dei veicoli e dei controller

L'inizializzazione dei veicoli e dei controller viene eseguita così:

```
for i in range(N):
    if i == 0:
    vehicles.append(Vehicle(True, Controller_standard(r
        , h, k1, k2)))
    else:
        vehicles.append(Vehicle(False,
            Controller_standard(r, h, k1, k2)))
    vehicles[i].states.append(VehicleState(-i, i, 0,
        velocity))
    if i != 0:
        previous_vehicle = vehicles[i - 1]
    else:
        previous_vehicle = None
    vehicles[i].controller.update_state_init(0,
        previous_vehicle, vehicles[i], acceleration[0],
        omega[0])
```

I veicoli vengono creati e a ciascuno di essi viene associato un controller. Le posizioni iniziali dei veicoli vengono assegnate in modo arbitrario: al primo veicolo viene attribuita la posizione (0,0), al secondo la posizione (-1,1), al ter-

zo (-2,2), e così via. Infine, viene eseguita la funzione *update_ state_ init(self, k, prec, vehicle, a, w)*, che inizializza il primo stato del controllore."

Capitolo 3

Simulazione del sistema e analisi dei risultati

In questo capitolo vengono presentati e analizzati i risultati ottenuti dalle simulazioni effettuate tramite il software sviluppato e descritto nel capitolo 2. I risultati delle simulazioni sono rappresentati attraverso diversi grafici che illustrano il comportamento dinamico dei veicoli all'interno del convoglio. In particolare, vengono analizzate le traiettorie percorse dai veicoli, la relazione tra il tempo e la posizione lungo l'asse x e lungo l'asse y. Infine, è stato calcolato l'errore tra la posizione effettiva di ciascun veicolo e quella del suo predecessore, un parametro cruciale per valutare la precisione e l'efficacia dell'algoritmo di controllo del platooning. Questi risultati offrono una visione chiara delle prestazioni del sistema simulato e costituiscono la base per le considerazioni e le conclusioni che verranno sviluppate nei capitoli successivi. In questo capitolo verrà esaminato anche il processo di generazione della traiettoria che i veicoli del convoglio devono seguire.

3.1 Parametri della simulazione

Come discusso e illustrato nel capitolo 1, sia la legge di controllo standard che quella estesa richiedono l'impostazione di determinati parametri per poter funzionare correttamente. Nelle simulazioni condotte per generare e analizzare i risultati sperimentali, a questi parametri sono stati assegnati i seguenti valori:

- Ai parametri **k1** e **k2**, i guadagni in ciclo chiuso del controllore, sono stati assegnati ad entrambi il valore di 2,5.
- alla distanza di arresto **r** dei veicoli è stato assegnato il valore di 1 m.
- al time-gap **h**, ovvero il tempo necessario ad un veicolo di raggiungere il predecessore viaggiando a velocità costante, è stato assegnato il valore 0,2 s.
- al tempo di campionamento **T** è stato assegnato il valore 0,01 s.
- per indicare la durata massima della simulazione è stato usato il parametro **T_max** ed è stata impostata una durata massima di 20 s.
- Il numero totale di steps temporale viene facilmente calcolato come T_max/T ed salvato nel parametro **num_steps**
- il numero di veicoli del convoglio viene indicato con il parametro **N**, impostato ad 5.

Per semplicità è stato impostata anche la velocità di ogni veicolo ad 5 m/s.

3.2 Generazione delle traiettorie

Le traiettorie vengono generate utilizzando due vettori distinti: uno rappresenta l'accelerazione che il veicolo deve mantenere in ogni istante temporale, mentre l'altro descrive la velocità angolare che il veicolo deve seguire durante la simulazione. La traiettoria risultante è specifica per il veicolo leader del convoglio, e gli altri veicoli, attraverso il sistema di controllo, devono cercare di replicare fedelmente la traiettoria del veicolo che li precede. I vettori di accelerazione e di velocità angolare saranno composti da un numero di elementi pari ad *num_steps*.
Nelle simulazioni prese in esame è stato utilizzato come vettore dell'accelerazione il vettore nullo; questo per un fattore di semplicità. In questo modo i veicoli del convoglio procederanno sempre a velocità costante, senza accelerazioni e decelerazioni. Il vettore della velocità angolare invece determina la forma della traiettoria. Le simulazioni effettuate si basano su due traiettorie specifiche:

1. Una traiettoria **circolare**: i veicoli iniziano muovendosi in linea retta per i primi 7 secondi, per poi passare a una traiettoria circolare nei successivi 13 secondi della simulazione. Questa traiettoria viene generata assegnando valori pari a zero al vettore della velocità angolare per i primi $7/T$ passi temporali, mentre nei restanti $13/T$ passi è stata impostata una velocità angolare di 0,5 rad/s. Il risultato è una traiettoria che segue inizialmente un percorso rettilineo, per poi evolversi in un moto circolare. Come si può osservare dalla figura 3.1

2. Una traiettoria **curvilinea**. Come mostrato nella figura 3.2, la traiettoria è stata generata utilizzando un vettore della velocità angolare strutturato come segue: per i primi $3/T$ passi temporali, il vettore con-

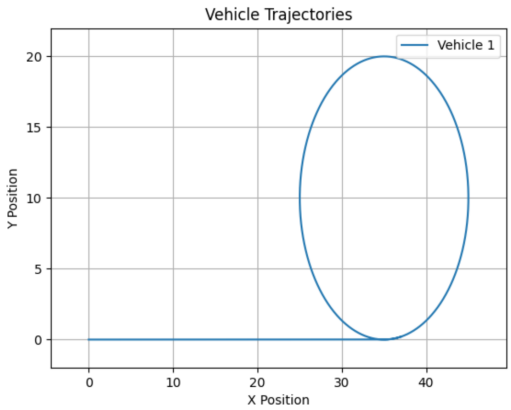

Figura 3.1: Traiettoria del veicolo leader del convoglio.

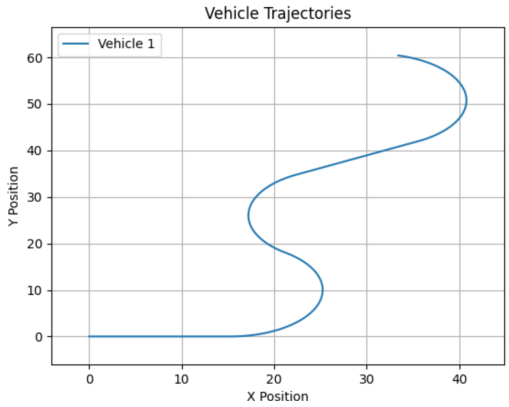

Figura 3.2: Traiettoria del veicolo leader del convoglio

tiene valori pari a 0, indicando un movimento rettilineo. Tra il passo $3/T$ e il passo $8/T$, è stata applicata una velocità angolare di 0,5 rad/s, che corrisponde a una curva a sinistra. Successivamente, fino al passo $12/T$, la velocità angolare è stata modificata a -0,5 rad/s, generando una curva a destra. Nei successivi 3 secondi di simulazione, il vettore torna a 0 per rappresentare un altro tratto rettilineo. Infine, negli ultimi 5 secondi di simulazione, la velocità angolare torna a 0,5 rad/s per produrre una curva a sinistra.

3.3 Simulazione delle traiettorie

In questa sezione analizzeremo le simulazioni del sistema. Come anticipato, consideriamo un convoglio di 5 veicoli, per un totale di 20 secondi di simulazione. Le traiettorie considerate sono quelle descritte nel paragrafo 3.2.

Per ogni traiettoria, verranno illustrati tre grafici distinti:

- Traiettoria del veicolo: Questo grafico mostra il percorso seguito dai veicoli nel piano XY, evidenziando la traiettoria percorsa durante la simulazione.
- Posizione lungo l'asse X rispetto al tempo: Questo grafico mette in risalto l'andamento della posizione lungo l'asse X in funzione del tempo, permettendo di analizzare come si evolve la coordinata X dei veicoli durante la simulazione.
- Posizione lungo l'asse Y rispetto al tempo: Analogamente al precedente, questo grafico mostra l'andamento della posizione lungo l'asse Y in funzione del tempo, evidenziando l'evoluzione della coordinata Y dei veicoli.

Infine, verrà mostrato anche un grafico tridimensionale che rappresenta la posizione dei veicoli in funzione delle coordinate X e Y e del tempo. Questo grafico fornisce una visione complessiva del movimento dei veicoli nello spazio e nel tempo, permettendo una comprensione approfondita della dinamica del sistema.

3.3.1 Simulazione con Look-ahead-based controller

Consideriamo per prima la **traiettoria circolare**.

In figura 3.3 è possibile osservare chiaramente il fenomeno del *taglio curva*,

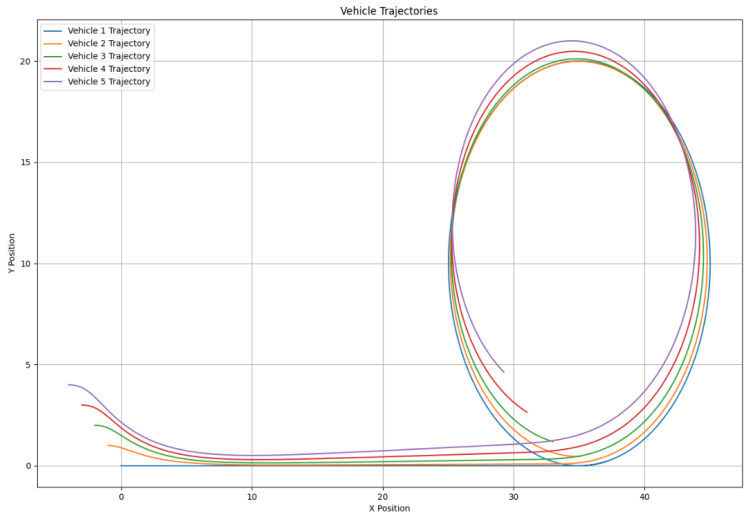

Figura 3.3: Traiettorie del convoglio di 5 veicoli con Look-ahead-based controller

descritto nel paragrafo 1.2. A partire dal secondo veicolo, fino al quinto, si nota che la traiettoria dei veicoli nel percorso circolare diverge rispetto a quella del veicolo leader. In particolare, i veicoli successivi iniziano la fase di curvatura in anticipo, generando una traiettoria leggermente differente. Questo problema diventa più pronunciato man mano che si considerano veicoli più lontani dal leader, poiché l'errore introdotto tende a propagarsi lungo il convoglio, amplificando la deviazione dalla traiettoria originale.

In figura 3.4 e in figura 3.5 possiamo osservare la posizione dei veicoli in funzione del tempo, in particolare in figura 3.4 si osserva la posizione lungo l'asse delle X, mentre in figura 3.5 si osserva la posizione dei veicoli lungo l'asse Y. Una visione tridimensionale della traiettoria dei veicoli in funzione del tempo è illustrata in figura 3.6.

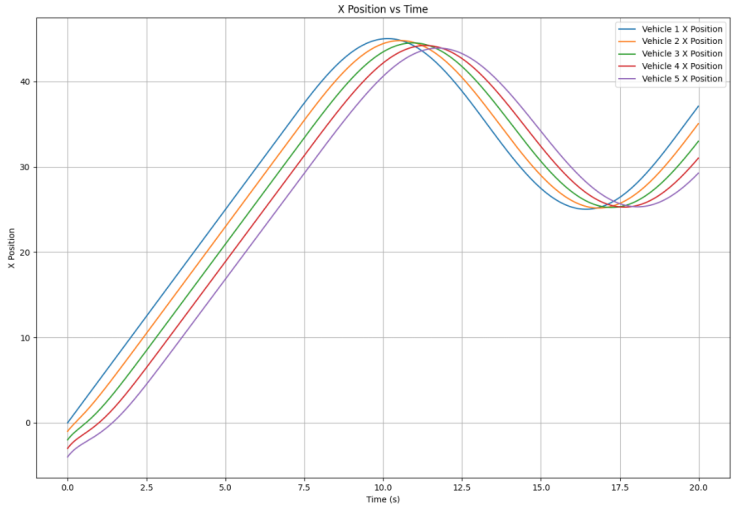

Figura 3.4: Posizione dei veicoli del convoglio lungo l'asse X in funzione del tempo.

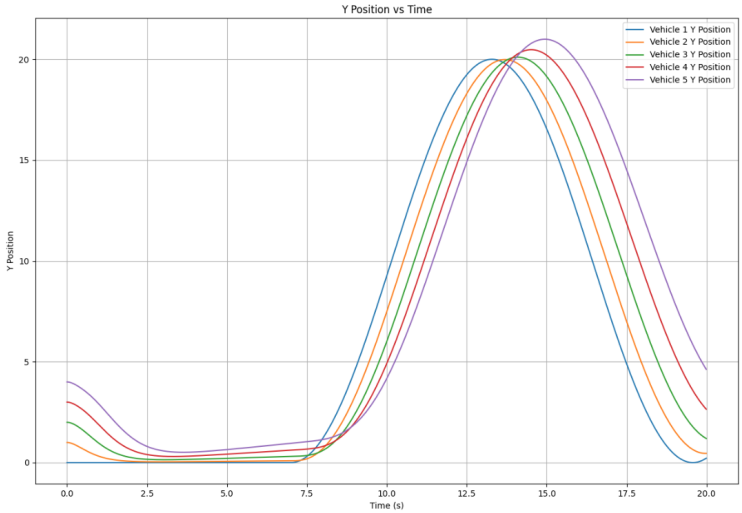

Figura 3.5: Posizione dei veicoli del convoglio lungo l'asse Y in funzione del tempo.

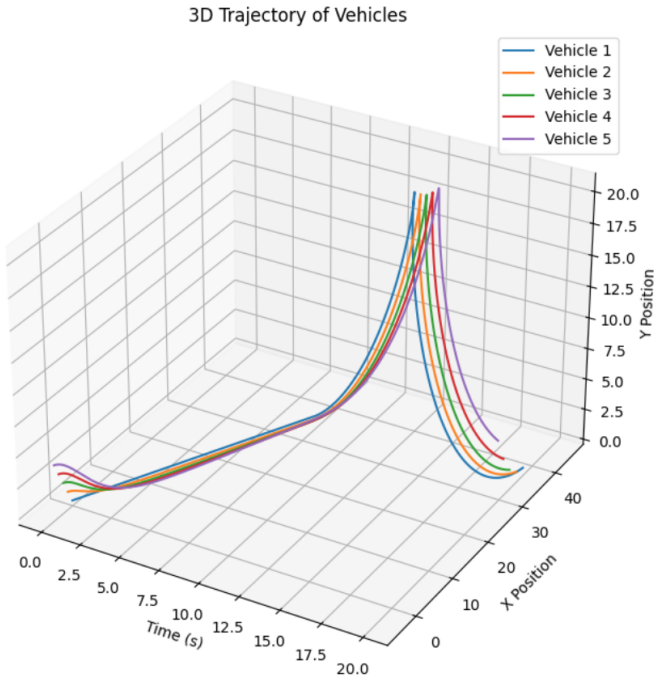

Figura 3.6: Grafico 3D delle traiettorie dei veicoli del convoglio in funzione dello spazio e del tempo.

Adesso consideriamo la traiettoria **curvilinea**

In figura 3.7 è possibile osservare le traiettorie dell'intero convoglio di veicoli,

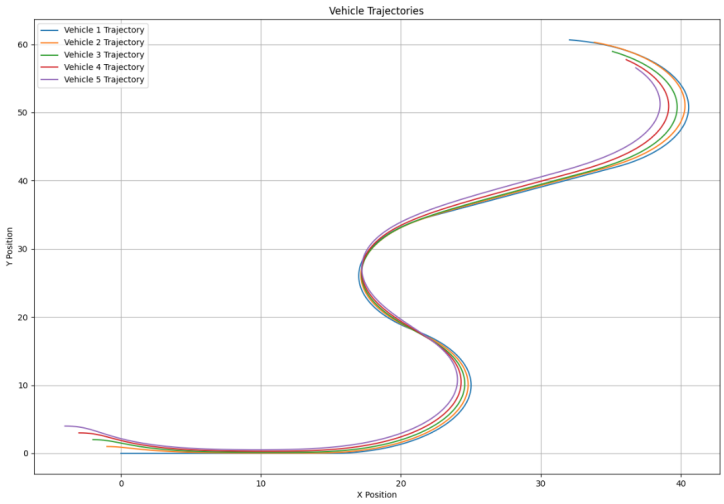

Figura 3.7: Traiettorie del convoglio di 5 veicoli con Look-ahead-based controller

anche come nel caso in figura 3.3, è possibile osservare il fenomeno del *taglio di curva*.

Nelle figure 3.8 e 3.9 è possibile osservare la posizione di ogni membro del convoglio in funzione del tempo.

Per finire in figura 3.10 è illustrata una visone tridimensionale delle traiettorie dell'intero convoglio in funzione dello spazio e del tempo.

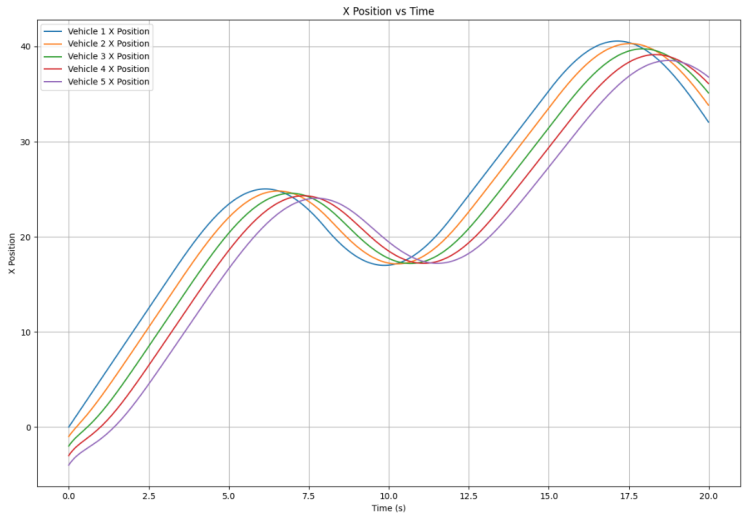

Figura 3.8: Posizione dei veicoli del convoglio lungo l'asse delle X in funzione del tempo

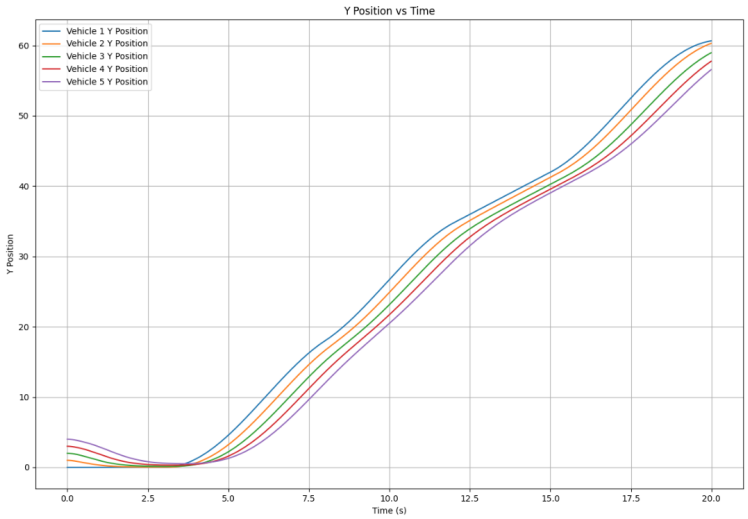

Figura 3.9: Posizione dei veicoli del convoglio lungo l'asse delle Y in funzione del tempo

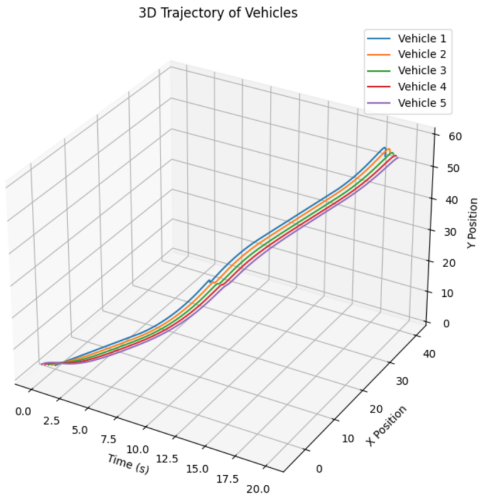

Figura 3.10: Grafico 3D delle traiettorie dei veicoli del convoglio in funzione dello spazio e del tempo

3.4 Calcolo dell'errore

Nella gestione del platooning, l'errore tra i veicoli rappresenta un parametro cruciale per valutare l'efficacia della legge di controllo applicata. Questo errore, definito come la differenza tra la posizione del veicolo attuale e quella del veicolo che lo precede, fornisce una misura diretta della capacità del sistema di mantenere una distanza di sicurezza e di rispondere alle variazioni dinamiche all'interno del convoglio. Un errore ridotto indica un controllo più preciso, essenziale per garantire la stabilità e l'efficienza complessiva del platooning. In questa sezione, verranno approfonditi i metodi di calcolo dell'errore di posizione e il loro ruolo nella valutazione delle prestazioni della legge di controllo.
Come già anticipato il calcolo dell'errore viene così calcolato:
$E_i(t) = \|P_i(t) - P_{i-1}(t - h - \frac{r}{v_i})\|$
dove: $P_i(t)$ è la posizione del veicolo i al tempo t, mentre la posizione del veicolo $P_{i-1}(t - h - \frac{r}{v_i})$ rappresenta la posizione del veicolo $i - 1$ al tempo $t - h - \frac{r}{v_i}$.
Questo termine è utilizzato per calcolare l'errore tra i due veicoli nel momento in cui entrambi dovrebbero trovarsi nello stesso punto. L'idea alla base di questa formula è quella di considerare il tempo necessario affinché il veicolo i raggiunga la posizione che il veicolo $i - 1$ ha occupato in precedenza, tenendo conto del ritardo temporale h e della distanza r che separa i due veicoli, rapportata alla velocità del veicolo v_i.

Il calcolo dell'errore viene effettuato in ogni istante temporale e i valori ottenuti vengono memorizzati in un vettore. Questo vettore, che raccoglie l'errore a ogni passo temporale, viene successivamente utilizzato per generare un grafico che mostra l'evoluzione dell'errore nel tempo, consentendo di

analizzare come varia l'accuratezza del sistema di controllo durante l'intero intervallo di osservazione.

Un altro parametro cruciale è la media dell'errore commesso dal singolo veicolo durante l'intera simulazione. Questa media, indicata con E_i, viene calcolata come segue:
$E_i = \sqrt{\sum_{t=1}^{T} \frac{1}{T} \|P_i(t) - P_{i-1}(t - h - \frac{r}{v_i})\|^2}$
Qui, T rappresenta il numero totale di istanti temporali considerati nella simulazione. Questo parametro fornisce una misura complessiva dell'accuratezza del controllo, riflettendo la deviazione media del veicolo i rispetto alla posizione attesa del veicolo precedente durante tutto il periodo di simulazione.

3.4.1 Implementazione del calcolo dell'errore

Di seguito è mostrato il codice per il calcolo dell'errore appena descritto.

```
def Error(vehicle, prec, T_max, T, h, v, r,vehicle_number):
    error = []
    times = np.arange(0, T_max, T)
    num_steps = len(times)

    Pos_vehicle = [(vehicle.states[i].x, vehicle.states[i].
        y) for i in range(num_steps)]

    # Calcolo della posizione del veicolo precedente al
        tempo t-h-r/v(t)
    Pos_prec = []
    for i in range(num_steps):
```

```
            t_minus_h_minus_v_r = i - int(h + (r/v[i])/T)
            if 0 <= t_minus_h_minus_v_r < len(prec.states):
                Pos_prec.append((prec.states[
                    t_minus_h_minus_v_r].x, prec.states[
                    t_minus_h_minus_v_r].y))
            else:
                Pos_prec.append((None, None))

    # Calcolo dell'errore tra le posizioni dei veicoli
    for i in range(num_steps):
        if Pos_prec[i] == (None, None):
            error.append(np.nan)  # Uso di nan per indicare
                valori non validi
        else:
            diff = np.array(Pos_vehicle[i]) - np.array(
                Pos_prec[i])
            error.append(np.linalg.norm(diff))



    # Plot dell'errore tra le posizioni dei veicoli
    plt.figure(figsize=(10, 6))
    plt.plot(times, error, label=f'Error between Vehicle {
        vehicle_number} and Vehicle {vehicle_number - 1}')
    plt.title(f'Error between Vehicle {vehicle_number} and
        Vehicle {vehicle_number -1} Positions over Time')
    plt.xlabel('Time (s)')
    plt.ylabel('Error (L2 Norm)')
    plt.legend()
    plt.grid(True)
```

```
    plt.show()

    return error
```

Questo codice oltre la calcolo dell'errore genera un grafico che mette in relazione l'errore tra i due veicoli ad ogni passo temporale.
Nel calcolo dell'errore, può emergere una particolarità rappresentata dal valore **NaN** (Not a Number). Questo valore viene utilizzato per indicare che il risultato del calcolo non è valido. In particolare, un valore **NaN** può apparire quando l'istante temporale $t - h - \frac{r}{v_i}$ non rientra nell'intervallo compreso tra 0 e $len(prec.states)$. In altre parole, si inseriscono valori **NaN** nel vettore dell'errore quando non è possibile calcolare l'errore per un determinato istante temporale, poiché la posizione del veicolo precedente al tempo $t - h - \frac{r}{v_i}$ non è disponibile. Questo accade quando il valore $t - h - \frac{r}{v_i}$ esce dai limiti temporali consentiti.

Passiamo ora a esaminare l'implementazione del calcolo dell'errore medio per ciascun veicolo.

```
#Calcolo dell'errore medio di ogni veicolo
Error_medio = []
for j in range(1, N):
    # Inizializza la lista 'sum' con zeri
    sum_errors = [0] * num_steps

    # Calcola la somma dei quadrati degli errori
    for i in range(num_steps):
        if np.isnan(Error_total[j - 1][i]):
            sum_errors[i] = sum_errors[i - 1] if i > 0
```

```
                else 0
        else:
            if i == 0:
                sum_errors[i] = Error_total[j - 1][i]
                    ** 2
            else:
                sum_errors[i] = sum_errors[i - 1] + (
                    Error_total[j - 1][i] ** 2)

    average_sum = sum_errors[num_steps - 1] / num_steps
    Error_medio.append(math.sqrt(average_sum))

    print(f"Errore medio tra i veicoli {j} e {j-1} : {
        Error_medio[j - 1]}")
```

Nell'implementazione del calcolo dell'errore medio per ogni veicolo, viene inizializzato un vettore chiamato *sum_errors* a 0. Questo vettore viene aggiornato a ogni istante temporale in due modi distinti:

- **Caso 1: Errore pari a NaN**

  Se l'errore calcolato per un determinato istante temporale è **NaN**, il valore di *sum_errors* rimane invariato e non viene effettuato alcun aggiornamento.

- **Caso 2: Errore valido**

  Se l'errore calcolato non è **NaN**, allora *sum_errors* viene aggiornato aggiungendo al valore precedente l'errore al quadrato. Nel caso del primo istante temporale, *sum_errors* viene semplicemente aggiornato con il valore dell'errore al quadrato.

Questo processo si ripete per ogni istante temporale, consentendo di accumulare progressivamente i valori necessari per calcolare la media dell'errore. Al termine del ciclo, nel vettore *sum_ errors[num_ steps - 1]* si trova la somma di tutti gli errori al quadrato. A questo punto, la somma viene divisa per il numero totale di istanti temporali *num_ steps*, e successivamente si calcola la radice quadrata del risultato. Questo fornisce il valore finale dell'errore medio per il veicolo in questione.

3.4.2 Grafico dell'errore

Di seguito sono riportati i grafici dell'errore relativi ad ogni veicolo. Consideriamo prima la **traiettoria circolare**, usando il **Look-ahead-based controller**:

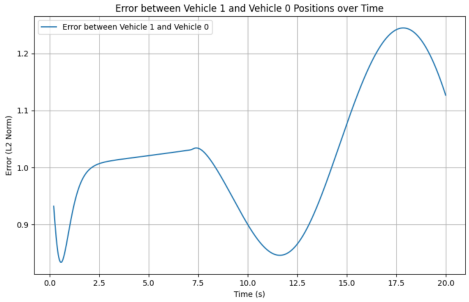

Figura 3.11: Errore tra il veicolo 1 e il veicolo 0, leader del convoglio

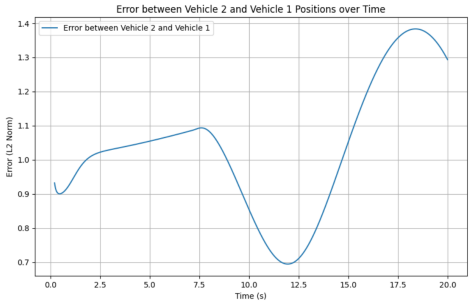

Figura 3.12: Errore tra il veicolo 2 e il veicolo 1

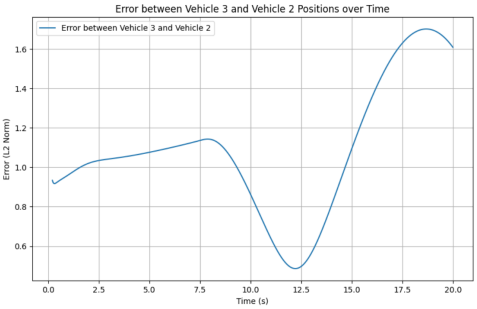

Figura 3.13: Errore tra il veicolo 3 e il veicolo 2

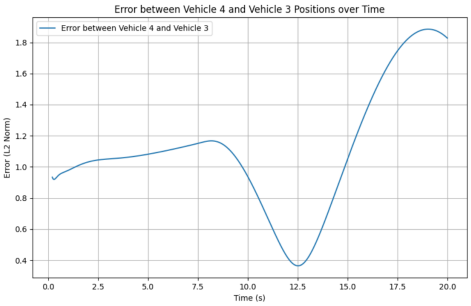

Figura 3.14: Errore tra il veicolo 4 e il veicolo 3

Figura 3.15: Grafici degli errori tra i veicoli in una traiettoria circolare

Durante la simulazione gli errori medi tra i veicoli sono i seguenti:

- **Tra i veicoli 1 e 0**: l'errore medio è **1.023 m**
- **Tra i veicoli 2 e 1**: l'errore medio è **1.047 m**
- **Tra i veicoli 3 e 2**: l'errore medio è **1.125 m**
- **Tra i veicoli 4 e 3**: l'errore medio è **1.167 m**

Vediamo adesso come appaiono i grafici dell'errore se consideriamo la **traiettoria curvilinea**, sempre usando il **Look-ahead-based controller**.

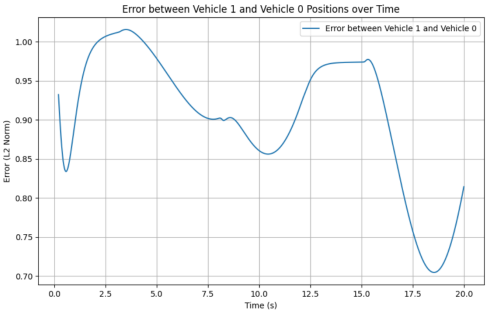

Figura 3.16: Errore tra il veicolo 1 e il veicolo 0, leader del convoglio

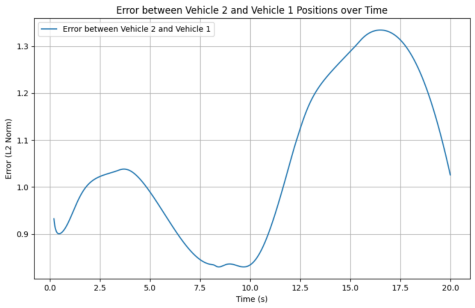

Figura 3.17: Errore tra il veicolo 2 e il veicolo 1

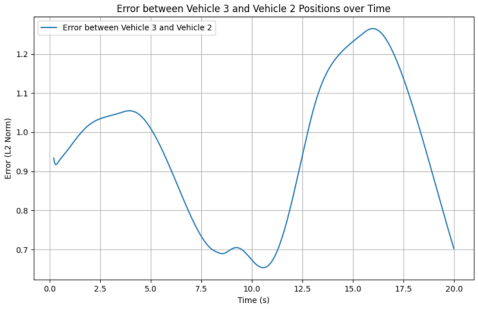

Figura 3.18: Errore tra il veicolo 3 e il veicolo 2

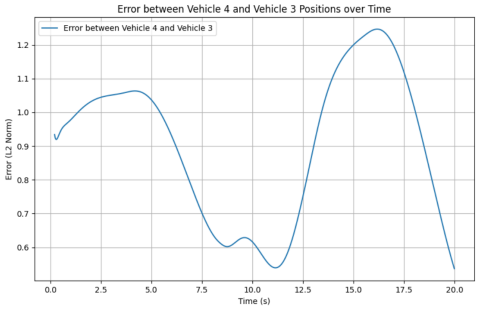

Figura 3.19: Errore tra il veicolo 4 e il veicolo 3

Figura 3.20: Grafici degli errori tra i veicoli in una traiettoria circolare

Durante la simulazione gli errori medi tra i veicoli sono i seguenti:

- **Tra i veicoli 1 e 0**: l'errore medio è **0.905 m**
- **Tra i veicoli 2 e 1**: l'errore medio è **1.063 m**
- **Tra i veicoli 3 e 2**: l'errore medio è **0.965 m**
- **Tra i veicoli 4 e 3**: l'errore medio è **0.925 m**

Dai dati analizzati, possiamo affermare che l'**errore** tra i veicoli del platooning **rimane pressoché costante**. Non si osservano differenze signifi-

cative nell'errore tra i veicoli posti alla fine del convoglio rispetto a quelli posizionati più avanti. In altre parole, l'errore non aumenta significativamente nei veicoli più arretrati del convoglio rispetto a quelli che si trovano più vicino alla testa del convoglio.
Questo comportamento dell'errore è stato riscontrato sia per traiettorie circolari che per traiettorie curvilinee. Indipendentemente dalla forma della traiettoria seguita, non vi è una tendenza evidente che suggerisca un incremento dell'errore nei veicoli situati alla fine del convoglio rispetto ai veicoli più vicini alla testa. Pertanto, l'errore si distribuisce uniformemente lungo il convoglio in entrambi i tipi di traiettoria.

Capitolo 4

Conclusioni

In questo lavoro, abbiamo sviluppato e analizzato una legge di controllo per il platooning di veicoli, con l'obiettivo di garantire la formazione di un convoglio coeso e sicuro, mantenendo la distanza interveicolare e la stabilità del sistema. I risultati ottenuti dimostrano che la legge di controllo proposta si comporta in maniera efficace nelle condizioni previste, riuscendo a mantenere la formazione desiderata del convoglio.

Tuttavia, è importante sottolineare che i risultati derivano da simulazioni eseguite in un ambiente privo di ritardi nelle comunicazioni tra i veicoli e senza rumore nelle misurazioni. Pertanto, gli esperimenti sono stati condotti in condizioni ideali, escludendo interferenze o disturbi esterni che potrebbero compromettere le prestazioni del sistema in contesti reali.

Durante le simulazioni, è emersa una limitazione significativa riguardante il comportamento del sistema nelle curve. In queste situazioni, i veicoli tendono a deviare leggermente dalla traiettoria ottimale a causa del cosiddetto "taglio di curva". Questo fenomeno è dovuto principalmente alla difficoltà

della legge di controllo nel gestire efficacemente le variazioni di angolo di sterzata dei veicoli che seguono, riducendo la precisione del convoglio nelle curve. Tale comportamento potrebbe avere impatti rilevanti in scenari di guida reali, specialmente in ambienti urbani o su strade con curve strette e frequenti.

Per il futuro, si prevede l'integrazione di soluzioni avanzate, come algoritmi di controllo più sofisticati per la gestione ottimale delle traiettorie in curva, o un migliore coordinamento tra velocità e sterzata all'interno del convoglio. Inoltre, lo sviluppo futuro potrebbe includere l'analisi di scenari più realistici, introducendo il ritardo nelle comunicazioni tra veicoli e considerando la presenza di rumore nelle misurazioni.

Questi miglioramenti, insieme a una gestione più accurata delle dinamiche dei veicoli nelle curve, potrebbero rafforzare ulteriormente la robustezza e l'affidabilità del sistema di controllo, avvicinandolo a un'implementazione su larga scala in condizioni reali.

Bibliografia

[1] Anggera Bayuwindra, Jeroen Ploeg, Erjen Lefeber, and Henk Nijmeijer. Combined longitudinal and lateral control of car-like vehicle platooning with extended look-ahead. *IEEE Transactions on Control Systems Technology*, 28:790–803, 2020.