



UNIVERSITÀ DI FIRENZE
DIPARTIMENTO DI INFORMATICA

Corso di Laurea Triennale in
Ingegneria Informatica

**Frontend per la simulazione
di platooning di veicoli**

Relatore

Giorgio Battistelli

Candidati

Federico Marra

Alberto Del Buono Paolini

Indice

1	Introduzione	1
2	Platooning di veicoli	4
2.1	Modello matematico del platooning	4
2.1.1	Equazioni in forma matriciale	5
2.1.2	Svolgimento equazioni a tempo continuo	6
2.1.3	Discretizzazione delle equazioni	7
2.2	Parametri	8
2.2.1	Time Headway (h)	8
2.2.2	Tau (τ)	8
2.2.3	Kp (k_p) e Kd (k_d)	8
2.3	L _p string stability	9
3	Simulazione del platooning	10
3.1	Implementazione delle equazioni	10
3.2	Rappresentazione globale delle equazioni nel software	14
3.3	Intervalli dei parametri di simulazione	15
4	Frontend	16
4.1	Impostazioni	17
4.2	Grafici	20
4.3	Scorciatoie da tastiera	23
4.4	Internazionalizzazione	24
4.5	Integrazione continua	26
5	Eperimenti	27
5.1	Modelli stabili	28
5.1.1	Parametri di default	30
5.1.2	Numero di auto (m)	31
5.1.3	Spazio tra le auto (d_i)	33
5.1.4	Ritardo di comunicazione	35
5.1.5	Time Headway (h)	39
5.1.6	Tau (τ)	42
5.1.7	Kp (k_p)	44
5.1.8	Kd (k_d)	46
5.1.9	Velocità (v_1) con parametri di default	48

5.2 Modelli instabili	52
5.2.1 Ritardo di comunicazione	53
5.2.2 Velocità (v_1) con parametri non default	55
6 Conclusioni	59
6.1 Conclusioni degli esperimenti	59
6.2 Applicazioni pratiche del platooning	60

1 Introduzione

La crescita del traffico stradale e la capacità limitata delle infrastrutture autostradali rappresentano sfide sempre più pressanti per le società moderne. L'incremento costante del numero di veicoli in circolazione ha portato a una saturazione sempre maggiore delle strade. Oltre a un aumento della durata degli spostamenti, con conseguenti ritardi, si osserva un incremento dello stress negli automobilisti e impatti negativi sul piano ambientale, dovuti alle emissioni generate dal traffico fermo. In questo contesto, la ricerca di soluzioni innovative e efficienti per migliorare la gestione del traffico stradale è diventata sempre più importante.

Tra le molteplici strategie proposte per affrontare questo problema, una delle più promettenti è il concetto di *platooning* di veicoli. Detto anche convo-gliamento o plotonamento, consiste nell'allineamento e nella gestione coordinata di più veicoli in movimento lungo una strada. Questa formazione segue un veicolo pilota cercando di mantenere una distanza costante prestabilita e ridotta tra di loro, minore della distanza di sicurezza; le vetture successive si adattano quindi in modo coordinato alle variazioni di velocità e direzione del veicolo di testa. Questo approccio offre numerosi vantaggi, tra cui una maggiore efficienza nel flusso del traffico, una riduzione della distanza tra i veicoli stessi e un miglioramento della sicurezza stradale in generale.

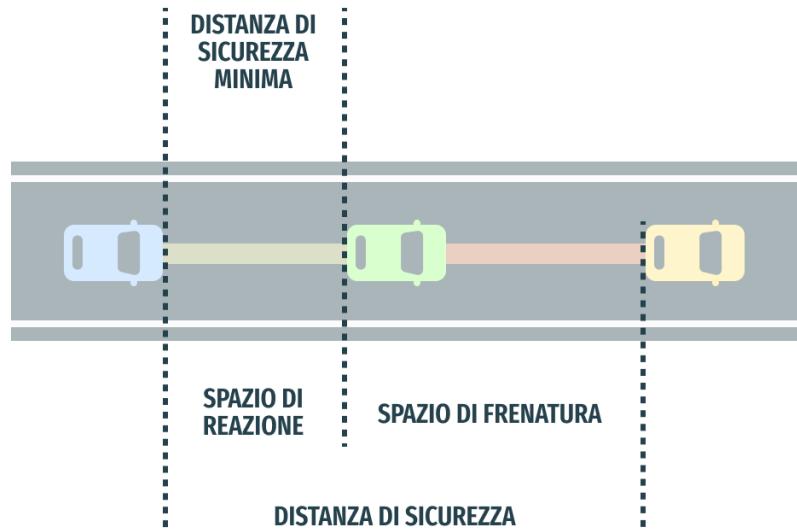


Figura 1: Distanza di sicurezza [PO21]

Introduzione

Tuttavia, la piena realizzazione del potenziale del *platooning* richiede l'adozione di sistemi avanzati di controllo e automazione dei veicoli. In particolare, è fondamentale sviluppare algoritmi e tecnologie in grado di coordinare in modo efficace e sicuro il movimento dei veicoli all'interno del plotone, garantendo contemporaneamente il rispetto delle norme stradali e la massima sicurezza per tutti gli utenti della strada.

A questo scopo una tecnologia che permette l'implementazione nel mondo reale del *platooning* è il controllo di crociera adattivo e cooperativo (CACC), che si presenta come soluzione molto promettente. Questo sistema utilizza dati provenienti dai veicoli stessi, oltre a informazioni ottenute da sensori come telecamere, radar e lidar (scanner laser), per regolare automaticamente la velocità e la distanza tra i veicoli all'interno del convoglio. Grazie alla sua capacità di scambio di dati tra veicoli, il CACC consente di mantenere intervalli di tempo di reazione significativamente inferiori rispetto ai sistemi di controllo di crociera adattivi tradizionali che, non accedendo ai dati dei veicoli circostanti necessitano di più tempo per decidere come e quale controllo applicare al veicolo. Dunque il CACC contribuisce così ad aumentare la capacità delle strade, ridurre il consumo di carburante e la sicurezza dei guidatori.

Bisogna vagliare il caso in cui ci sia bisogno di frenare bruscamente, se c'è una persona alla guida c'è da tenere conto del tempo di reazione, calcolato mediamente in un range fra 1s e 1.1s che corrisponde però a una distanza maggiore proporzionalmente alla velocità a cui si sta viaggiando come mostrato in figura 3. Considerando dunque anche lo spazio di frenata, la distanza di sicurezza nelle auto con guidatore convenzionale è mostrata nelle figure 1 e 2.

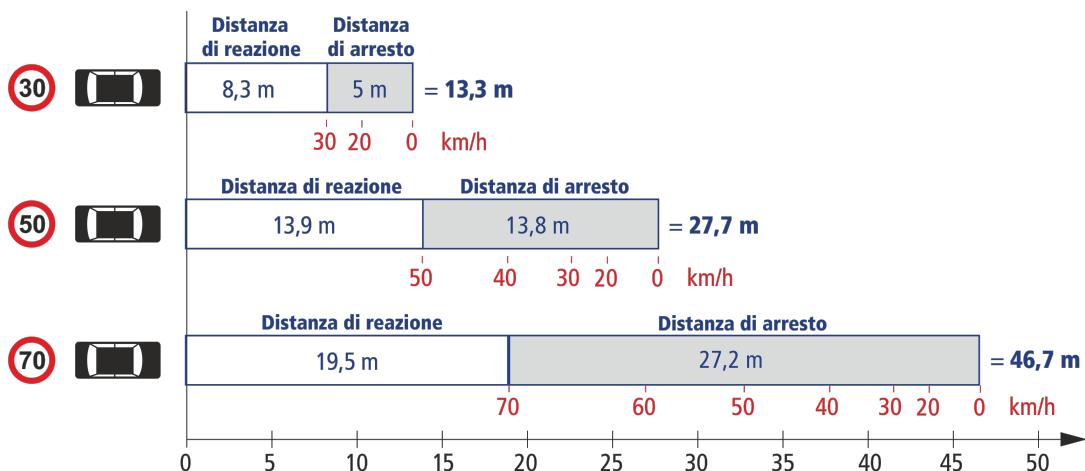


Figura 2: Componenti distanza di sicurezza [AQ16]

SPAZIO DI REAZIONE

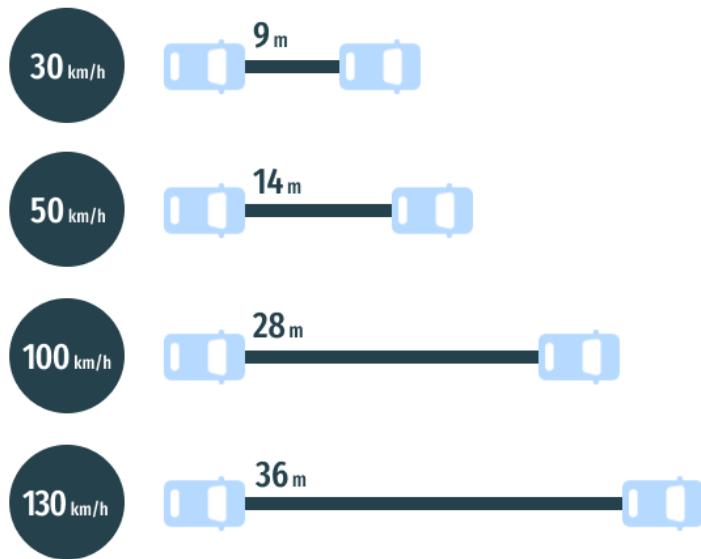


Figura 3: Spazio di reazione [PO21]

Nonostante i suoi evidenti vantaggi, l'implementazione efficace del CACC e quindi dei sistemi di *platooning* richiede la progettazione e lo sviluppo di un software sofisticato che permetta una valutazione delle prestazioni di tali sistemi, qui il nostro progetto entra in gioco. Ci proponiamo di sviluppare un *frontend* innovativo che permetta la simulazione del fenomeno del *platooning* di veicoli su un'unica direzione longitudinale. Ci siamo concentrati sull'implementazione del modello fisico e del sistema di controllo più vicini possibile a quelli reali per fini meramente divulgativi, che permettano lo sviluppo e l'avanzamento di questa tecnologia che mira a massimizzare l'efficienza del traffico stradale, garantendo al contempo la massima sicurezza per tutti gli utenti stradali.

2 Platooning di veicoli

Per l'implementazione del sistema di controllo utilizziamo il modello descritto nell'articolo [PWN14].

2.1 Modello matematico del platooning

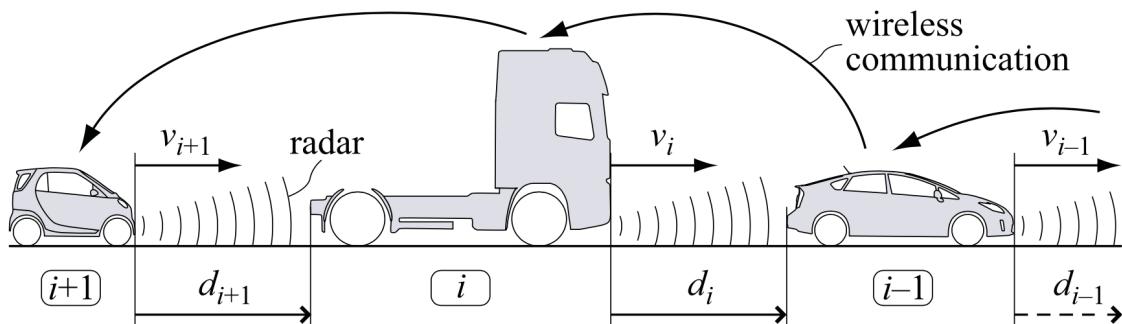


Figura 4: *Platoon* di veicoli equipaggiati con CACC

Nel *platooning* si considera un convoglio di m vetture, come schematizzato nella Figura 4, dove d_i rappresenta la distanza tra il veicolo i e il veicolo precedente $i - 1$, e v_i rappresenta la sua velocità. L'obiettivo di ciascuna vettura è seguire il veicolo precedente a una distanza desiderata $d_{r,i}$ secondo l'equazione:

$$d_{r,i}(t) = r_i + h \cdot v_i(t), \quad i \in S_m \quad (1)$$

Nell'equazione (1) h è indicato come *time headway* o anche tempo di separazione e r_i è la distanza di fermo, ovvero la distanza che si vorrebbe mantenere tra le vetture di in uno stato di arresto. $S_m = \{i \in \mathbb{N} \mid 1 \leq i \leq m\}$ rappresenta l'insieme di tutti i veicoli in un convoglio di lunghezza $m \in \mathbb{N}$. L'equazione di spaziamento (1) è progettata per migliorare la stabilità della stringa e la sicurezza. Si assume un convoglio omogeneo, quindi h è lo stesso per tutti i veicoli i . Definita $d_i(t)$ come la distanza fra i veicoli i e $i - 1$:

$$d_i(t) = q_{i-1}(t) - q_i(t) - L_i \quad (2)$$

dove q_i che rappresenta la posizione del paraurti posteriore del veicolo i e L_i la sua lunghezza. Riprendiamo dall'articolo [PWN14] l'equazione (3) nella quale l'errore di spaziamento $e_i(t)$ è definito come segue:

$$e_i(t) = d_i(t) - d_{r,i}(t) = (q_{i-1}(t) - q_i(t) - L_i) - (r_i + h \cdot v_i(t)) \quad (3)$$

Il problema di controllo comprende due requisiti: l'obiettivo di seguire il veicolo precedente e il requisito di stabilità di stringa spiegato successivamente nella sezione [2.3]. Rappresentiamo dunque in x_i le componenti del veicolo i a un certo stato t , per convenzione scriveremo:

$$x_i = \begin{bmatrix} e_i \\ v_i \\ a_i \\ u_i \end{bmatrix} \quad (4)$$

Il sistema di controllo oltre alle sue componenti ha bisogno di recuperare anche quelle della macchina che la precede, rappresentata da x_{i-1} . Il sistema avrà dunque l'evoluzione come descritto nell'equazione (5).

$$\dot{x}_i = A_0 x_i + A_1 x_{i-1} \quad (5)$$

2.1.1 Equazioni in forma matriciale

Dall'articolo [PWN14] traiamo l'equazione in forma matriciale della legge di controllo applicate al vettore \dot{x}_i :

$$\begin{bmatrix} \dot{e}_i \\ \dot{v}_i \\ \dot{a}_i \\ \dot{u}_i \end{bmatrix} = \begin{bmatrix} 0 & -1 & -h & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{\tau} & \frac{1}{\tau} \\ \frac{k_p}{h} & -\frac{k_d}{h} & -k_d - \frac{k_{dd}(\tau-h)}{h\tau} & -\frac{k_{dd}h+\tau}{h\tau} \end{bmatrix} \cdot \begin{bmatrix} e_i \\ v_i \\ a_i \\ u_i \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{k_d}{h} & \frac{k_{dd}}{h} & \frac{1}{h} \end{bmatrix} \cdot \begin{bmatrix} e_{i-1} \\ v_{i-1} \\ a_{i-1} \\ u_{i-1} \end{bmatrix} \quad (6)$$

Come suggerito dall'articolo [PWN14] poniamo $k_{dd} = 0$ dunque il sistema di evoluzione in forma matriciale diventa (svolgendo $-\frac{\tau}{h\tau} = -\frac{1}{h}$):

$$\begin{bmatrix} \dot{e}_i \\ \dot{v}_i \\ \dot{a}_i \\ \dot{u}_i \end{bmatrix} = \begin{bmatrix} 0 & -1 & -h & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{\tau} & \frac{1}{\tau} \\ \frac{k_p}{h} & -\frac{k_d}{h} & -k_d & -\frac{1}{h} \end{bmatrix} \cdot \begin{bmatrix} e_i \\ v_i \\ a_i \\ u_i \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{k_d}{h} & 0 & \frac{1}{h} \end{bmatrix} \cdot \begin{bmatrix} e_{i-1} \\ v_{i-1} \\ a_{i-1} \\ u_{i-1} \end{bmatrix} \quad (7)$$

2.1.2 Svolgimento equazioni a tempo continuo

Svolgiamo dunque il sistema descritto dall'equazione (7) nella quale si trascu-
ra il fattore k_{dd} , ponendolo a 0; otteniamo quindi le componenti di \dot{x}_i in funzione
delle variazioni delle seguenti quantità:

L'errore:

$$\dot{e}_i = -v_i - ha_i + v_{i-1} \quad (8)$$

La velocità:

$$\dot{v}_i = a_i \quad (9)$$

L'accelerazione:

$$\dot{a}_i = -\frac{a_i}{\tau} + \frac{u_i}{\tau} = \frac{-a_i + u_i}{\tau} \quad (10)$$

Il controllo:

$$\begin{aligned} \dot{u}_i &= \frac{k_p}{h} e_i - \frac{k_d}{h} v_i - k_d a_i - \frac{u_i}{h} + \frac{k_d}{h} v_{i-1} + \frac{u_{i-1}}{h} = \\ &= \frac{k_p e_i - k_d v_i - u_i + k_d v_{i-1} + u_{i-1}}{h} - k_d a_i \end{aligned} \quad (11)$$

Così otteniamo le componenti delle distanze x_i e, nella prossima sezione, an-
dremo a discretizzarle utilizzando il metodo di Eulero.

2.1.3 Discretizzazione delle equazioni

Per far sì che le equazioni dinamiche siano implementabili in un sistema informatico che per sua natura richiede un campionamento, si pone un tempo (o una frequenza di campionamento) e le equazioni (8), (9), (10) e (11) diventano da continue, discrete:

L'errore:

$$\Delta e_i = -v_i - ha_i + v_{i-1} \quad (12)$$

La velocità:

$$\Delta v_i = a_i \quad (13)$$

L'accelerazione:

$$\Delta a_i = -\frac{a_i}{\tau} + \frac{u_i}{\tau} = \frac{-a_i + u_i}{\tau} \quad (14)$$

Il controllo:

$$\begin{aligned} \Delta u_i &= \frac{k_p}{h} e_i - \frac{k_d}{h} v_i - k_d a_i - \frac{u_i}{h} + \frac{k_d}{h} v_{i-1} + \frac{u_{i-1}}{h} = \\ &= \frac{k_p e_i - k_d v_i - u_i + k_d v_{i-1} + u_{i-1}}{h} - k_d a_i \end{aligned} \quad (15)$$

Le componenti del veicolo i allo stato t nel sistema saranno descritte dall'equazione $x_i(t) = x_i(t-1) + \Delta x_i(t-1) \cdot T_s$ (con T_s tempo di campionamento):

$$e_i(t) = e_i(t-1) + \Delta e_i(t-1) \cdot T_s \quad (16a)$$

$$v_i(t) = v_i(t-1) + \Delta v_i(t-1) \cdot T_s \quad (16b)$$

$$a_i(t) = a_i(t-1) + \Delta a_i(t-1) \cdot T_s \quad (16c)$$

$$u_i(t) = u_i(t-1) + \Delta u_i(t-1) \cdot T_s \quad (16d)$$

Comporremo così il nostro modello a partire dall'equazione (3), svolgendola per ottenere la distanza i -esima ricaviamo:

$$d_i(t) = e_i(t) + d_{r,i}(t) = e_i(t) + r_i + h \cdot v_i(t) \quad (17)$$

dove:

- $e_i(t)$ è ottenuto dall'equazione (16a).
- r_i è la *standstill distance* detta anche distanza di fermo, decisa a priori.
- h è il *time headway* anche questo deciso a priori e uguale per tutti i veicoli.
- $v_i(t)$ è ottenuto dall'equazione (16b)

2.2 Parametri

Di seguito andiamo a spiegare i parametri della legge di evoluzione (7), cioè le varie impostazioni del modello di *platooning* da noi successivamente implementate nel *frontend*, rendendole modificabili dall'utente.

2.2.1 Time Headway (h)

Il *time headway*, detto anche tempo di separazione svolge una vera e propria funzione di sicurezza: rappresenta l'intervallo di tempo tra un veicolo e il suo predecessore, in altre parole indica la distanza temporale desiderata tra i veicoli all'interno del convoglio. È mostrato come varia negli esperimenti nella sezione [5.1.5].

2.2.2 Tau (τ)

Tau rappresenta una costante di tempo che descrive la dinamica interna al veicolo, pertanto indica quanto rapidamente il veicolo risponde al controllo dell'accelerazione u_i . Poiché si assume l'omogeneità del plotone, tutti i veicoli avranno il medesimo valore di τ . È mostrato come varia negli esperimenti nella sezione [5.1.6].

2.2.3 Kp (k_p) e Kd (k_d)

Kp e Kd rappresentano i parametri del controllore adottato, utilizzati per calcolare il segnale di controllo u_i dell'equazione (11) e Δu_i dell'equazione (15). k_p e k_d sono rispettivamente i coefficienti proporzionale e derivativo nel controllo PID (proporzionale-integrale-derivativo). Questi parametri influenzano la risposta del controllore agli errori di distanza e velocità del veicolo rispetto al suo obiettivo. È mostrato come varia negli esperimenti nelle sezioni [5.1.7] e [5.1.8].

Nell'articolo [PWN14], a pagina 791 si fa riferimento a dei valori per cui è conservata la stabilità. Questi sono i valori che abbiamo impostato di default nella simulazione, come riportato qui sotto e nella sezione [5.1.1].

Time Headway (h)	Tau (τ)	Kp (k_p)	Kd (k_d)
0.5s	0.1	0.2	0.7

2.3 L_p string stability

Nel contesto della stabilità di Lyapunov, la stabilità delle stringhe rappresenta una forma specifica di analisi della stabilità applicata a sistemi di veicoli interconnessi, quindi comunemente usata nello studio del *platooning*.

La teoria della stabilità di Lyapunov fornisce un quadro per analizzare la stabilità dei sistemi dinamici, concentrando sul comportamento di una funzione di Lyapunov. Nella stabilità delle stringhe, l'analisi di Lyapunov viene impiegata per valutare se le distanze e le velocità relative dei veicoli in un convoglio rimangono costanti nel tempo. Ecco come si analizza la stabilità delle stringhe:

1. **Funzione di Lyapunov:** Nell'analisi della stabilità delle stringhe, viene spesso utilizzata una funzione di Lyapunov per quantificare le proprietà di stabilità del sistema del convoglio. Questa funzione misura la deviazione delle distanze e delle velocità inter-veicolo dai valori desiderati.
2. **Criterio di Stabilità:** La stabilità delle stringhe si raggiunge se la funzione di Lyapunov diminuisce nel tempo, indicando che le deviazioni dalle distanze e velocità inter-veicolo desiderate si riducono man mano che il tempo passa senza che eventuali disturbi vengano amplificati al crescere del numero di veicoli. Questo corrisponde al criterio di stabilità di Lyapunov, dove un sistema è considerato stabile se esiste una funzione di Lyapunov che diminuisce lungo le traiettorie del sistema.
3. **Analisi di Lyapunov:** Analizzando la dinamica del sistema del convoglio con le funzioni di Lyapunov, si possono determinare le condizioni in cui il convoglio rimane stabile e progettare algoritmi di controllo per soddisfare tali condizioni, come quello descritto in precedenza.

In generale, la stabilità delle stringhe coinvolge l'applicazione delle tecniche di analisi di Lyapunov per valutare e garantire la stabilità dei convogli o plotoni di veicoli autonomi, con l'obiettivo di mantenere distanze e velocità dei veicoli consistenti nel tempo e proteggerli da eventuali disturbi esterni.

Viene inoltre notato in [PWN14] a pagina 788 che in assenza di ritardi di comunicazione, il *platoon* è stabile asintoticamente per ogni *time headway* $h > 0$ e ogni $k_p, k_d > 0$ e $k_d > k_p\tau$. Chiaramente l'assenza di ritardo di comunicazione è una condizione ideale e non realistica, per questo abbiamo aggiunto un parametro nel *frontend* con cui modificarlo.

3 Simulazione del platooning

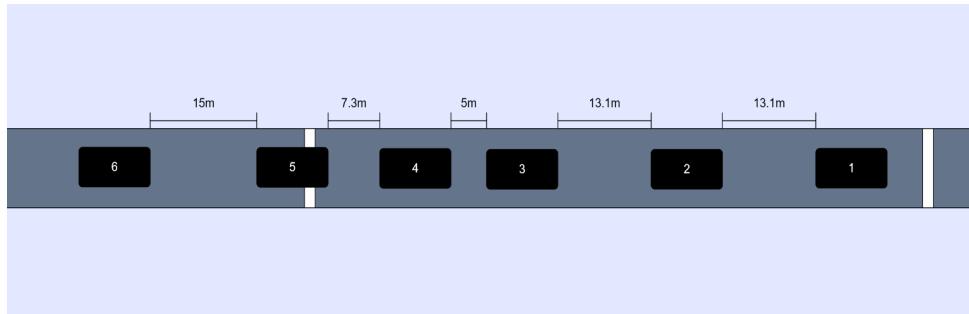


Figura 5: Visualizzazione grafica della simulazione.

3.1 Implementazione delle equazioni

Le equazioni teoriche ricavate nelle sezione [2.1.2] e [2.1.3], utili ai fini della simulazione, sono implementate nella funzione `draw` del *canvas* di *p5.js*. Come inizializzazione dei valori del primo veicolo (o pilota) poniamo:

$$e_0 = 0 \quad (18a)$$

$$a_0 = \frac{v_0 - v_0^-}{T_s} \quad (18b)$$

$$v_0 += a_0 \cdot T_s = \frac{a_0}{F_s} \quad (18c)$$

$$u_0 = a_0 \quad (18d)$$

con $T_s = \frac{1}{F_s}$ e viceversa $F_s = \frac{1}{T_s}$. Nell'implementazione v_0 e v_0^- sono i valori di velocità corrispondenti all'iterazione corrente e precedente di `draw`. Qui di seguito è possibile osservare l'aggiornamento dei parametri del veicolo pilota.

Linee 257–264 di *P5Canvas*

```
prevV[0] = velocity[0];
prevU[0] = controlU[0];
error[0] = 0;
acceleration[0] = leadingCarChart[(leadingCarChartIndex + 1)
    leadingCarChart.length].velocity - leadingCarChart[leadingCarChartIndex].velocity;
velocity[0] += acceleration[0] / FS;
controlU[0] = acceleration[0];
```

Nella pagina successiva segue l'inizializzazione della distanza di fermo r_i dall'equazione (2) come `standstillDistance`.

Linea 266 di *P5Canvas*

```
const standstillDistance = carSpacing * 10 + CAR_WIDTH;
```

Dopo di che il ciclo `for` itera per compiere il calcolo delle varie componenti di ogni veicolo, escluso il primo, a ogni *frame*.

Linee 269–333 di *P5Canvas*

```
for (let i = 1; i < carNumber; i++) {
    ...
}
```

Dunque nell'iterazione del veicolo $i \neq 0$ al tempo t salviamo prima i valori di $e_i(t - 1)$ e $u_i(t - 1)$ in `prevE` e `prevA`.

Linee 271–272 di *P5Canvas*

```
const prevE = error[i];
const prevA = acceleration[i];
```

Ora applichiamo le equazioni (12) con (16a), (13) con (16b), (14) con (16c) e (15) con (16d):

$$e_i(t) += -v_i(t - 1) - ha_i(t - 1) + v_{i-1}(t) \quad (19a)$$

$$v_i(t) += a_i(t - 1) \quad (19b)$$

$$a_i(t) += -\frac{a_i(t - 1)}{\tau} + \frac{u_i(t - 1)}{\tau} \quad (19c)$$

$$\begin{aligned} u_i(t) += \frac{k_p}{h}e_i(t - 1) - \frac{k_d}{h}v_i(t - 1) - k_da_i(t - 1) - \frac{u_i(t - 1)}{h} + \\ + \frac{k_d}{h}v_{i-1}(t - 1) + \frac{u_{i-1}(t - 1)}{h} \end{aligned} \quad (19d)$$

riscrivibili come:

$$e_i(t) += v_{i-1}(t) - v_i(t - 1) - ha_i(t - 1) \quad (20a)$$

$$v_i(t) += a_i(t - 1) \quad (20b)$$

$$a_i(t) += \frac{u_i(t - 1) - a_i(t - 1)}{\tau} \quad (20c)$$

$$\begin{aligned} u_i(t) += \frac{k_p e_i(t - 1) - k_d v_i(t - 1) + k_d v_{i-1}(t - 1)}{h} + \\ + \frac{u_{i-1}(t - 1) - u_i(t - 1)}{h} - k_da_i(t - 1) \end{aligned} \quad (20d)$$

Simulazione del platooning

Traduciamo quindi le equazioni (20) nel codice, ricordando che per la simulazione $F_s = 30$ equivale al *frame rate* di $30fps$ (*frame* per secondo).

Linee 282–300 di *P5Canvas*

```
error[i] += (prevV[i-1] - prevV[i] - timeHeadway * prevA) / FS;
velocity[i] += prevA / FS;
acceleration[i] += ((prevU[i] - prevA) / tau) / FS;
controlU[i] += ((kp * prevE - kd * prevV[i] - prevU[i] + kd * prevV[i-1] + prevU[i-1]) /
timeHeadway - kd * prevA) / FS;
```

Calcolate le componenti di x_i usiamo l'equazione (17) con $d_{r,i}$ definita come `desiredDistance`, r_i come `standstillDistance` e d_i come `d`.

Linee 303–304 di *P5Canvas*

```
let desiredDistance = standstillDistance + velocity[i] * timeHeadway;
let d: number = error[i] + desiredDistance;
```

Successivamente sorge una problematica: mettere un limite superiore allo spazio percorribile da un veicolo in un *frame*. Nei termini della nostra modellizzazione di un sistema che simula ciò che accadrebbe in una situazione reale, dobbiamo trattare il problema di non avere accelerazione infinita istantanea. La soluzione è implementata forzando uno spazio massimo percorribile per *frame*, rintracciabile e modificabile nel codice come `maxStep`.

Linee 306–323 di *P5Canvas*

```
const maxStep = 0.75;
const prevDistance = Math.abs(carPoints[i] - carPoints[i-1]);

if (prevDistance - d > maxStep) {
    d = prevDistance - maxStep;
    for (let j = i; j < carNumber; j++) {
        carPoints[j] += maxStep;
    }
} else if (d - prevDistance > maxStep) {
    if (leadingCarChart[leadingCarChartIndex].velocity !== 0) {
        d = prevDistance + maxStep;
        for (let j = i; j < carNumber; j++) {
            carPoints[j] -= maxStep;
        }
    } else {
        d = prevDistance;
    }
}
```

Abbiamo anche introdotto dei limiti superiori e inferiori per le velocità, rispettivamente a 35 m/s e -10 m/s , considerando possibile la retromarcia.

Linee 285–290 di *P5Canvas*

```
if (velocity[i] > 50) {
    velocity[i] = 50;
} else if (velocity[i] < -10) {
    velocity[i] = -10;
}
```

Oltre al `maxStep`, andiamo anche a definire un ulteriore vincolo con `maxAcceleration`, limitando lo *step* dell'accelerazione ad ogni *frame*.

Linee 274–279 di *P5Canvas*

```
const maxAcceleration: number = 5;
if (prevA > maxAcceleration) {
    prevA = maxAcceleration;
} else if (prevA < maxAcceleration * -1) {
    prevA = maxAcceleration * -1;
}
```

Dopo aver calcolato la variabile `d` che indica la distanza tra i veicoli i e $i - 1$, andiamo ad aggiornare il valore assoluto della posizione del veicolo i -esimo, contenuto in `carPoints`.

Linea 326 di *P5Canvas*

```
carPoints[i] = carPoints[i - 1] - d;
```

Infine aggiorniamo i valori `prevV` e `prevU`, preparandoli per la prossima iterazione del ciclo di simulazione. Questo aggiornamento avviene solo ogni `velocityFrameDelay` iterazioni, implementando così il parametro del ritardo di comunicazione tra le vetture.

Linee 329–332 di *P5Canvas*

```
if (p5.frameCount % velocityFrameDelay === 0) {
    prevV[i] = velocity[i];
    prevU[i] = controlU[i];
}
```

3.2 Rappresentazione globale delle equazioni nel software

Funzione *draw* di *P5Canvas*

```
// Inizializzazione per il primo veicolo
error[0] = 0;
acceleration[0] = leadingCarChart[(leadingCarChartIndex + 1) %
    leadingCarChart.length].velocity - leadingCarChart[leadingCarChartIndex].velocity;
velocity[0] += acceleration[0] / FS;
controlU[0] = acceleration[0];
...

// Ciclo di aggiornamento degli altri veicoli
for (let i = 1; i < carNumber; i++) {
    // I valori prevE, prevV, prevA e prevU corrispondono all'istante di tempo t-1
    const prevE = error[i];
    const prevA = acceleration[i];

    // Limita la variazione massima che l'accelerazione
    // di una vettura può compiere in un frame
    const maxAcceleration: number = 5;
    if (prevA > maxAcceleration) {
        prevA = maxAcceleration;
    } ...

    // Calcola Δei, Δvi, Δai, Δui e discretizza secondo Eulero
    error[i] += (prevV[i-1] - prevV[i] - timeHeadway * prevA) / FS;
    ...

    // Limita a un massimo e minimo la velocità della vettura
    if (velocity[i] > 50) {
        velocity[i] = 50;
    } else if (velocity[i] < -10) {
        velocity[i] = -10;
    }

    // di = ei + ri (standstillDistance) + vi * h (timeHeadway)
    const desiredDistance = standstillDistance + velocity[i] * timeHeadway;
    let d: number = error[i] + desiredDistance;

    // Limita la distanza massima che un veicolo può coprire in un frame
    const maxStep = 0.75;
    ...

    // Aggiorna la posizione del veicolo sul canvas
    carPoints[i] = carPoints[i - 1] - d;

    // Aggiorna velocità e controllo precedenti con delay
    if (p5.frameCount % velocityFrameDelay === 0) {
        prevV[i] = velocity[i];
        prevU[i] = controlU[i];
    }
}
```

3.3 Intervalli dei parametri di simulazione

Come successivamente spiegato nella sezione [4.1], abbiamo implementato il pannello di impostazioni selezionando degli intervalli coerenti per i parametri delle equazioni descritte in precedenza. I parametri hanno quindi i seguenti gradini (*step*), minimi e massimi:

Parametro	step	min	max
N° auto	1	2	10
Distanza Target	0.1m	2m	20m
Ritardo	0.1s	0s	4s
Time Headway (h)	0.01s	0.01s	2s
Tau (τ)	0.01s	0.01s	2s
Kp (k_p)	0s	0.01s	2s
Kd (k_d)	0s	0.01s	2s
Velocità a t_i	1 m/s	0 m/s	35 m/s

Tabella 1: Intervalli per i vari parametri

Tutto ciò è implementato nel codice in SettingSliver immettendo i valori della tabella 1 nei vari input presenti.

Linee 100–106 di SettingSliver

```
<input
    type="range"
    step="0.01"
    min="0.01"
    max="2"
    value={timeHeadway}
/>
```

Per la distanza iniziale invece abbiamo optato per un valore generato randomicamente tra 1m e 15m con almeno 5m di differenza dalla distanza target, come mostrato nel blocco di codice sottostante.

Linee 456–459 di P5Canvas

```
let initDistance = desiredDistance;
while (Math.abs(initDistance - desiredDistance) <= 5)
    initDistance = Math.random() * 14 + 1;
```

4 Frontend

Per sviluppare il progetto abbiamo usato *Next.js*¹, framework *fullstack* per lo sviluppo di applicazioni web, accompagnato da *TypeScript* per rendere lo sviluppo più attendibile, aggiungendo sicurezza rispetto ai tipi per tutto il codice.

Tutta la cronologia dello sviluppo è disponibile su *Github*² e il frontend è accessibile pubblicamente su platooning-simulation.vercel.app³. Abbiamo optato per ospitare l'app su Vercel⁴ dato che offre integrazione/distribuzione continua (*CI/CD*), come discuteremo in una sezione successiva. Il rilascio della nostra applicazione non è tuttavia limitato solo a questa piattaforma ma è anche compatibile con progetti AWS *Amplify* e *Netlify*.

La visualizzazione della simulazione vera e propria è implementata usando *p5.js*⁵, libreria che ci permette di gestire più efficientemente il rendering ad alto *frame rate* sul *canvas* HTML e che rende disponibile una *API* più ergonomica per interagire con esso.

Per gestire gli stili dell'interfaccia utente abbiamo usato *TailwindCSS*⁶, un framework CSS che ci ha permesso di non dividere il codice di rendering dal codice di *styling* (usando il meccanismo delle classi CSS), rendendo lo sviluppo dei componenti *UI* più veloce, rintracciabile e standardizzato.

Esempio di classi *TailwindCSS*

```
<div className="p-1 text-white transition-all duration-300 rounded-md
    cursor-pointer select-none bg-slate-800 hover:bg-slate-300
    hover:text-slate-800">
    <InfoIcon />
</div>
```

Abbiamo reso disponibili anche varie scorciatoie da tastiera per eseguire diverse azioni nell'interfaccia utente, ad esempio SPAZIO mette in pausa o fa partire la simulazione, come spiegato successivamente nella sezione [4.3].

¹ <https://nextjs.org>

² <https://github.com/albbus-stack/platooning-simulation>

³ <https://platooning-simulation.vercel.app>

⁴ <https://vercel.com>

⁵ <https://p5js.org>

⁶ <https://tailwindcss.com>

4.1 Impostazioni

La gestione dello stato delle impostazioni di simulazione avviene sfruttando il *contesto React*⁷. Il meccanismo di funzionamento del contesto è molto simile a quello del pattern *Provider/Consumer* e si divide quindi in due parti:

- *Provider*: viene instanziato un componente che racchiude tutti i componenti di cui esso deve gestire lo stato, permettendo interazioni tra loro.
- *Consumer*: per consumare (leggere) oppure aggiornare lo stato di un *Provider* viene usato l'*hook* `useContext`⁸ che espone queste funzionalità ad ogni componente sottostante al *Provider*.

Nel nostro caso il DataProvider contiene tutte le varie impostazioni della simulazione che possono essere modificate dal componente DataProvider (la sezione che contiene i selettori orizzontali per ogni opzione ed il grafico della velocità del primo veicolo); lo stato di queste impostazioni viene in seguito letto dal *canvas p5* per aggiornare la simulazione stessa. Questo meccanismo ci permette quindi di sincronizzare facilmente lo stato tra il componente di modifica e quello di visualizzazione.

Sono disponibili le seguenti impostazioni di simulazione (visibili in figura 6): *Numero di veicoli*, *Distanza obiettivo tra di loro*, *Distanza temporale tra di loro*, *Delay di comunicazione* ed alcuni parametri del modello di *platooning* precedentemente discusso nelle sezioni [2.2.2] e [2.2.3].

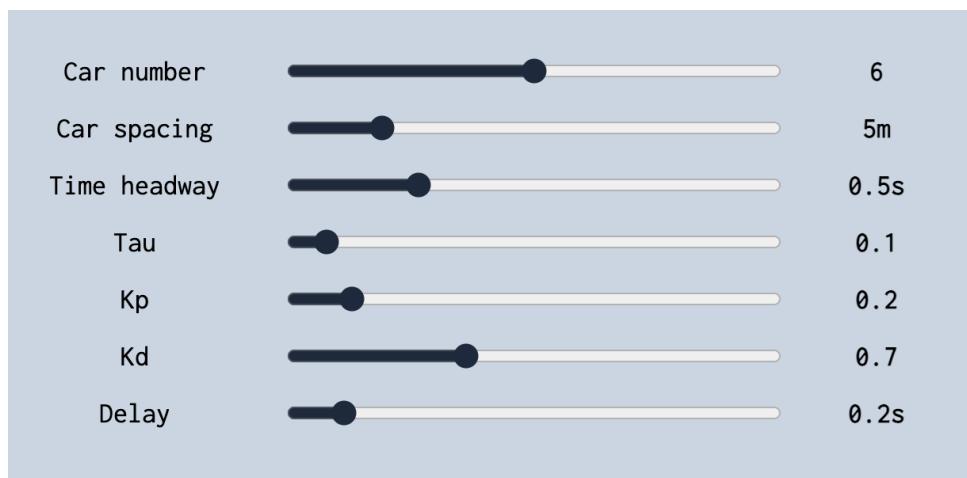


Figura 6: Varie impostazioni numeriche per la simulazione.

⁷ <https://react.dev/learn/passing-data-deeply-with-context>

⁸ <https://react.dev/reference/react/useContext>

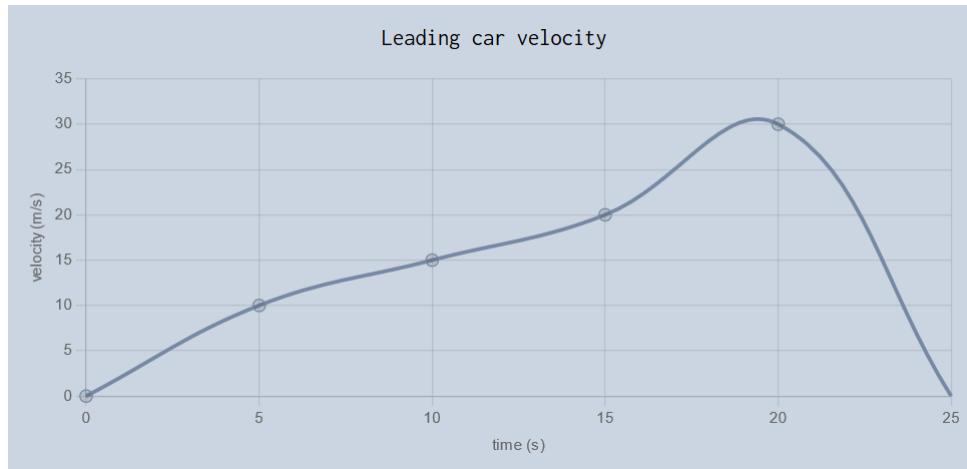


Figura 7: Grafico per impostare l'andamento del primo veicolo del convoglio nel tempo.

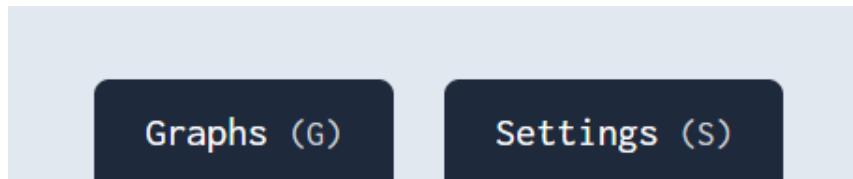


Figura 8: Interfaccia utente per l'apertura dei pannelli (*Sliver*), con sopra annotate le scorciatoie da tastiera.

La velocità del primo veicolo del convoglio è controllabile tramite un grafico interattivo, in figura 7, in cui sono modificabili cinque punti da $t = 0$ a $t = 20$. Il veicolo pilota segue in maniera periodica questo grafico delle velocità con accelerazione uniforme. Come descritto nella sezione successiva, è stato scelto *ChartJS* per produrre anche questo grafico interattivo.

Le impostazioni per la simulazione ed i grafici risultanti sono contenute in due pannelli separati, accessibili tramite due bottoni nell'interfaccia utente (visibili in figura 8) oppure usando delle scorciatoie da tastiera: G per aprire il pannello dei grafici (GraphSliver) e S per aprire il pannello delle impostazioni (SettingSliver), come meglio descritte nella sezione [4.3].

Dopo aver configurato i parametri desiderati, è possibile avviare la simulazione per visualizzare il comportamento del veicolo in base alle scelte effettuate. L'uso di grafici e impostazioni personalizzabili permette un'analisi interattiva del modello di *platooning* da parte dell'utente.

Codice allegato

Gestione dello stato delle impostazioni con *DataProvider*

```

/* DataProvider.tsx */
export const DataProvider: React.FC<DataProviderProps> = ({ children }) => {
  const [carNumber, setCarNumber] = useState(6);
  const [carSpacing, setCarSpacing] = useState(5.0);
  const [timeHeadway, setTimeHeadway] = useState(0.5);
  const [tau, setTau] = useState(0.1);
  const [kp, setKp] = useState(0.2);
  const [kd, setKd] = useState(0.7);
  const [velocityFrameDelay, setVelocityFrameDelay] = useState(VELOCITY_DELAY);
  const [leadingCarChart, setLeadingCarChart] = useState<GraphPoints[]>(
    [0, 1, 2, 3, 4].map((i) => {
      return {
        time: i,
        velocity: i * 2 + 2,
      };
    })
  );
  ...
}

/* SettingsSliver.tsx */
const SettingsSliver: React.FC = () => {
  const { carNumber, setCarNumber, carSpacing, setCarSpacing, ... } =
    useContext(DataContext);
  ...
}

/* P5Canvas.tsx */
<NextReactP5Wrapper
  sketch={sketch}
  carSpacing={carSpacingSetting}
  carNumber={carNumberSetting}
  ...
>

```

Le varie impostazioni della simulazione sono contenute dentro *DataProvider* e i loro stati sono inizializzati con dei valori di default, inclusi i punti del grafico della velocità del primo veicolo. In seguito *SettingSliver* utilizza i *getter* e *setter* esposti da questo *provider* per permettere all'utente di modificare le impostazioni interattivamente attraverso gli *sliders*.

Infine nel *canvas p5* viene letto lo stato del contesto e passato all'istanza sottostante di *p5.js* attraverso i *props* del componente *NextReactP5Wrapper* che rende la simulazione consapevole del cambiamento di qualsiasi di questi, in maniera reattiva.

4.2 Grafici

La visualizzazione dei grafici è implementata usando *ChartJS*⁹, libreria molto popolare grazie alla performance del *canvas* HTML e alla sua interazione stretta con *React* (attraverso la libreria *wrapper react-chartjs-2*¹⁰).

Come suggerito precedentemente, la gestione dei dati usati per la produzione dei grafici è sempre affidata al *contexto React*, in particolare al componente *DataProvider* che espone *getter* e *setter* relativi ai dati di distanza e velocità tra i veicoli. Questo approccio ci consente di gestire in modo efficiente e organizzato tutti i dati relativi alla simulazione, garantendo una facile integrazione con il resto dell'applicazione.

La simulazione *p5* stessa esegue a *30 fps* e viene campionata ogni quarto di secondo (*250ms*) per raccogliere questi dati di distanza e velocità. Quando viene aggiunto un nuovo *DataPoint*, i grafici contenuti in *GraphSliver* si aggiornano, mostrando dinamicamente i dati raccolti, permettendo quindi agli utenti di monitorare l'evoluzione della simulazione in tempo reale.

Nell'interfaccia utente è possibile visualizzare il grafico della distanza tra due veicoli e la velocità specifica di un veicolo; questi sono selezionabili da due *dropdown* presenti sopra ai grafici corrispondenti.

È possibile osservare un esempio della visualizzazione di questa sezione in figura 9 e 10.

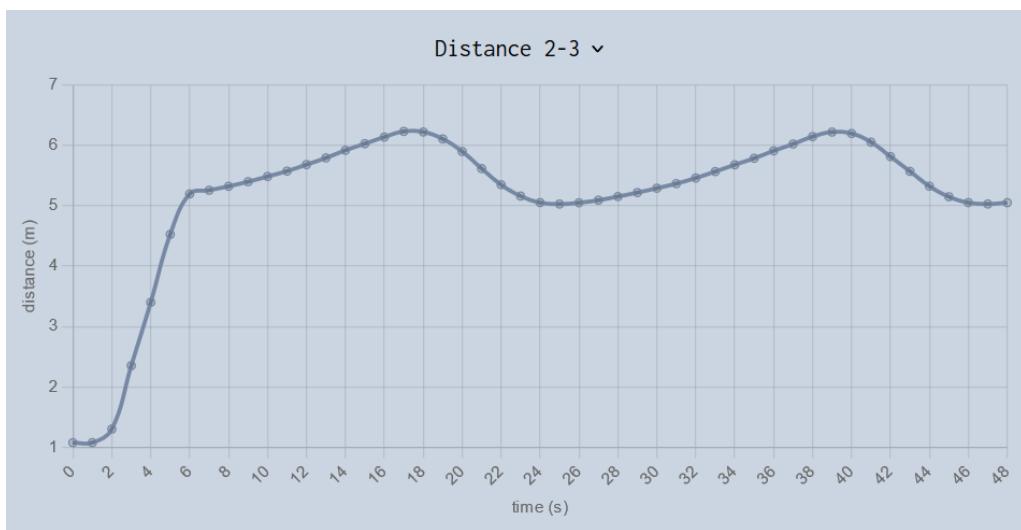


Figura 9: Grafico della distanza tra due veicoli del convoglio.

⁹ <https://www.chartjs.org>

¹⁰ <https://react-chartjs-2.js.org>

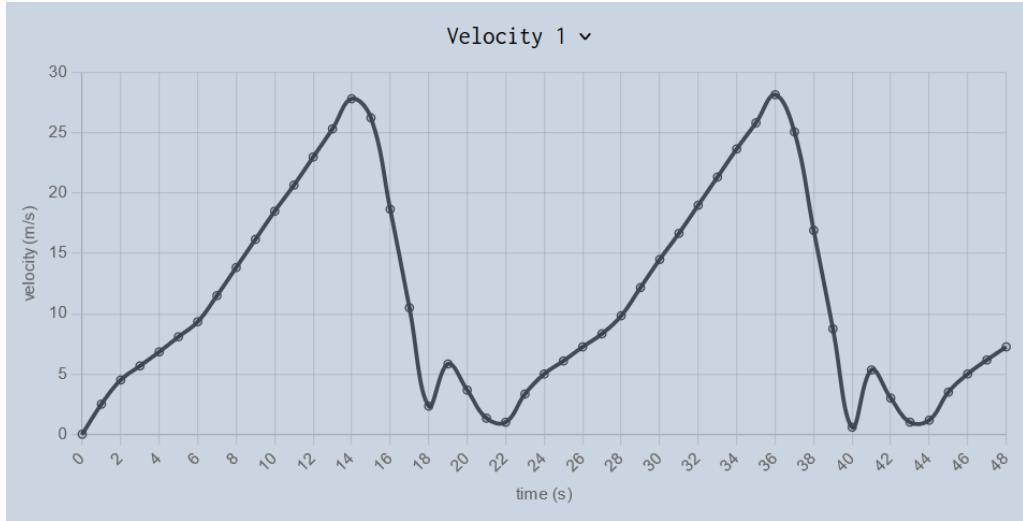


Figura 10: Grafico della velocità di un veicolo del convoglio.

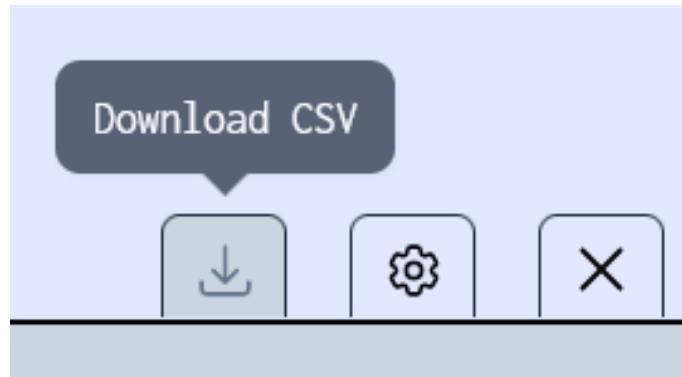


Figura 11: Bottone di esportazione dei dati della simulazione in csv.

Un aspetto utile della nostra implementazione è la possibilità di esportare i dati raccolti in formato csv con la seguente struttura: carNumber, time(s), distance(m), velocity(m/s). Dove time(s) è l'istante di tempo e distance si riferisce alla distanza tra il veicolo carNumber e carNumber+1. Per l'ultimo veicolo questa distanza sarà sempre considerata 0 come default dato che non c'è nessun veicolo successivo ad esso.

Il download in formato csv consente agli utenti di analizzare i dati della simulazione in dettaglio al di fuori dell'applicazione, utilizzando strumenti esterni o elaborando i risultati in altri contesti. In figura 11 è visibile l'interfaccia utente per questa funzione.

Codice allegato

Gestione dello stato dei grafici con *DataProvider*

```
/* DataProvider.tsx */
export const DataProvider: React.FC<DataProviderProps> = ({ children }) => {
  const [graphData, setGraphData] = useState<{}[][]> as {
    distance: number;
    velocity: number;
    time: number;
  }[][];
  ...
}

/* GraphSliver.tsx */
const GraphSliver: React.FC = () => {
  const [distanceChartIndex, setDistanceChartIndex] = useState(0);
  const [velocityChartIndex, setVelocityChartIndex] = useState(0);

  const { graphData } = useContext(DataContext);
  ...
}

/* P5Canvas.tsx */
const { setGraphData } = useContext(DataContext);

intervalRef = setInterval(() => {
  timeTick++;
  setGraphData((car) =>
    car.map((prev, i) => [
      ...prev,
      { time: timeTick, distance: distance[i], velocity: velocity[i] },
    ])
  );
}, UPDATE_INTERVAL);
```

Come si può osservare sopra, *DataProvider* contiene i dati di distanza e velocità di ogni veicolo per ogni istante di tempo. Questo stato è letto da *GraphSliver* che ha due variabili di stato locali contenenti gli indici dei grafici di distanza e velocità attualmente selezionati dall'utente nella sezione corrispondente.

Lo stato del *DataProvider* viene aggiornato dentro un intervallo (cioè una *API* per ripetere l'esecuzione di un blocco di codice periodicamente) nel *canvas p5* che contiene i dati della simulazione in esecuzione. Nel nostro caso l'intervallo esegue ogni `UPDATE_INTERVAL` millisecondi, costante definita precedentemente nel file ad un quarto di secondo; per ottenere dati in uscita con frequenza di campionamento più alta è sufficiente modificare questo valore.

4.3 Scorciatoie da tastiera

Abbiamo aggiunto varie scorciatoie da tastiera per interagire con l'interfaccia utente più rapidamente. Segue un elenco di queste *shortcuts*:

- SPAZIO avvia o interrompe la simulazione.
- R ripristina la simulazione alle impostazioni predefinite.
- S apre la sezione per la modifica delle impostazioni (SettingsSliver).
- G apre la sezione di visualizzazione dei grafici (GraphSliver).
- D scarica i dati in csv della simulazione eseguita.
- ↑ e ↓ ingrandiscono/diminuiscono la dimensione della Sliver.
- ← e → ciclano tra Sliver delle impostazioni e dei grafici.
- ESC chiude la Sliver.

Codice allegato

Gestione delle scorciatoie da tastiera in *P5Canvas.tsx*

```
useEffect(() => {
  const onKeyDown = (e: KeyboardEvent) => {
    switch (e.key) {
      // Start/stop the simulation
      case " ":
        togglePlay();
        break;

      // Reset the simulation
      case "r":
        resetCanvas(window.innerWidth, carNumberSetting, carSpacingSetting);
        break;
      ...
    }
  }

  document.addEventListener("keydown", onKeyDown);
  return () => document.removeEventListener("keydown", onKeyDown);
}, [togglePlay])
```

Il codice fornito gestisce le scorciatoie per SPAZIO e R utilizzando `useEffect`. Con questo *hook* viene registrato un gestore per gli eventi `keydown` quando il componente viene montato. Questo gestore di eventi controlla quale tasto è stato premuto attraverso l'oggetto `KeyboardEvent` e reagisce di conseguenza. Oltre a questo `useEffect` ritorna una funzione di *cleanup* dove viene rilasciato il gestore delle scorciatoie.

4.4 Internazionalizzazione

Per rendere questo progetto fruibile da utenti che non parlano l'inglese abbiamo deciso di implementare un sistema di internazionalizzazione che include le seguenti lingue: *en, it, fr, es, de, dk, nl, pt, ru, sa, in, cn, jp e kr*.

Questa funzionalità è stata raggiunta usando le *Dynamic Routes*¹¹; queste consistono nel raccogliere il codice di rendering per varie *routes* (percorsi del sito) dentro un solo componente. Nel nostro caso ci permettono di cambiare il linguaggio selezionato semplicemente rimandando al percorso /*locale* corrispondente (dove *locale* è una delle stringhe di localizzazione sopra elencate). Per la lingua di default, cioè l'inglese, il percorso usato è / senza nessuna specificazione di lingua ulteriore, come vediamo in figura 12.

La gestione vera e propria delle varie traduzioni è stata affidata a *ParaglideJS*¹², un nuovo e promettente framework di internazionalizzazione. Questa libreria gestisce lo stato locale del client riguardante la lingua attualmente selezionata ed espone una semplice *API* per leggerlo e modificarlo, tutto mantenendo completamente la *type safety*.

Le stringhe di traduzione sono scritte in dei file json e possiamo aggiungere altre lingue semplicemente scrivendone uno nuovo e ricompilando il *bundle* di *ParaglideJS* (affinché tutte le stringhe siano controllate ed esportate staticamente prima della *build* del progetto).

Per mostrare le bandiere corrispondenti ad ognuna delle lingue utilizzate abbiamo usato degli svg resi disponibili dalla repository *country-flags*¹³.



Figura 12: Percorsi dinamici corrispondenti ad inglese (/), italiano (/it) e francese (/fr).

¹¹ <https://nextjs.org/docs/pages/building-your-application/routing/dynamic-routes>

¹² <https://inlang.com/m/gerre34r/library-inlang-paraglideJs>

¹³ <https://github.com/hampusborgos/country-flags/tree/main/svg>

Codice allegato

Utilizzo di *ParaglideJS* per l'internazionalizzazione

```

/* pages/[locale]/index.tsx */
import { AvailableLanguageTag, availableLanguageTags, setLanguageTag } from
  "../../src/paraglide/runtime";
import { useRouter } from "next/router";

const Home: NextPage = () => {
  const router = useRouter();
  const locale = router.query.locale as AvailableLanguageTag ?? "en";

  if (availableLanguageTags.includes(locale)) {
    setLanguageTag(locale);
  } else if (router.query.locale) {
    return <PageNotFound />;
  }

  return <HomePage />;
};

/* translations/it.json */
{
  "$schema": "https://inlang.com/schema/inlang-message-format",
  "title": "Simulazione di platooning",
  "graphs": "Grafici",
  "settings": "Impostazioni", ...
}

/* package.json */
{
  "name": "platooning-simulation",
  "scripts": {
    "lang": "paraglide-js compile --project ./project.inlang.json",
  }, ...
}

```

Come si può osservare, i percorsi dinamici sono gestiti attraverso l'*hook* `useRouter`¹⁴ che raccoglie la stringa corrispondente alla *route* visitata nell'attributo `query.locale`. Se viene fornita una stringa vuota il *type cast* la assegnerà al valore della *locale* di default (`en`), altrimenti se viene fornita una stringa di localizzazione non valida verrà renderizzata una *pagina 404* (pagina non trovata; contenente un link alla pagina principale).

Come discusso in precedenza le stringhe delle traduzioni sono contenute in dei file json e dobbiamo compilare i messaggi di *ParaglideJS* ogni volta che queste vengono aggiornate (attraverso lo *script* eseguibile con `pnpm lang`).

¹⁴ <https://nextjs.org/docs/pages/api-reference/functions/use-router>

4.5 Integrazione continua

Per garantire un flusso di sviluppo fluido e automatizzato, abbiamo configurato un sistema di integrazione continua utilizzando *Vercel* e la sua integrazione *GitHub*¹⁵.

Vercel, una piattaforma di hosting specializzata nel *deploy* di applicazioni *React* e *Next.js*, offre un'integrazione continua semplice ed efficace: ogni volta che viene eseguito un *commit* o viene aperta una *pull request* sul *branch* principale della nostra repository viene avviato automaticamente un processo di *build* e distribuzione dell'applicazione.

Questo processo assicura che ogni modifica apportata al codice venga immediatamente testata e poi resa disponibile. Inoltre, *Vercel* offre la funzionalità dei *deployment* di *preview*, con la possibilità di creare anteprime delle modifiche non ancora pubblicate al pubblico, consentendo un controllo prima che queste vengano distribuite ufficialmente.

Codice allegato

Integrazione continua con *Vercel*

```
/* package.json */
{
  "name": "platooning-simulation",
  "scripts": {
    "lint": "next lint",
    "build": "pnpm lint && pnpm lang && pnpm next build",
  },
}
```

Questo *snippet* indica come vengono gestiti i passaggi di *build* dell'applicazione durante il processo di integrazione continua:

- `pnpm lint`: prima di avviare il processo di *build*, viene eseguito il comando di *linting* per assicurarsi che il codice sia conforme agli standard definiti.
- `pnpm lang`: vengono poi compilate le traduzioni dell'applicazione per renderle disponibili al *runtime* di *ParaglideJS* come discusso in precedenza.
- `pnpm next build`: infine viene avviato il processo di *build* ed esportazione dell'applicazione utilizzando *Next.js*.

¹⁵ <https://vercel.com/docs/deployments/git/vercel-for-github>

5 Esperimenti

I grafici sottostanti sono stati generati con uno script *Python* a partire da dei csv esportati dall’interfaccia web, tutte queste risorse sono disponibili nella cartella experiments sulla repository *Github*¹⁶ divise in cartelle relative al parametro testato.

L’interfaccia web permette infatti di eseguire una simulazione di test con durata di 40s, cioè includendo due cicli del grafico delle velocità del primo veicolo, da 20s ciascuno, ottenuti interpolando i 5 punti definiti dall’utente (uno ogni 4s). Questa funzione è esposta all’utente tramite una scorciatoia da tastiera (E) che, in seguito alla simulazione, scarica automaticamente il csv relativo rendendo più semplice e consistente il processo di esportazione dei dati.

Codice allegato

Generazione dei grafici con *Matplotlib*

```
def plot_and_save(output_filename, data, ylabel):
    for car_index, car_data in data.items():
        if ylabel == 'distance' and car_index == len(data.items()) - 1:
            continue
        plt.plot(car_data['time'], car_data[ylabel], marker='o', linestyle='-' ,
                 label=f'Auto {car_index + 1}' if ylabel == 'velocity'
                 else f'Distanza {car_index + 1}-{car_index + 2}')

    plt.xlabel('Tempo (s)')
    plt.ylabel('Velocità (m/s)' if ylabel == 'velocity' else 'Distanza (m)')
    plt.legend()
    plt.savefig(output_filename)
```

Prima dell’esecuzione di questo script, i dati esportati in csv vengono caricati in cartelle diverse a seconda del parametro che è stato variato prima dell’esperimento (rispetto a quelli di default riportati nella sezione [5.1.1]).

È stato usato *Matplotlib*¹⁷ per generare grafici delle velocità e delle distanze tra i veicoli della carovana e salvarle su vari file nominati coerentemente, dopo aver estratto i dati dai csv corrispondenti. Questi grafici sono particolarmente utili per comparare quanto influiscono le variazioni dei vari parametri sul modello usato; oltre a questo ci danno uno strumento di analisi e visualizzazione delle relazioni di velocità e distanza tra tutti i veicoli del *platoon*.

¹⁶ <https://github.com/albbus-stack/platooning-simulation/tree/main/experiments>

¹⁷ <https://matplotlib.org/>

5.1 Modelli stabili

Come spiegato nella sezione [2.2.3], sono stati usati i parametri di default suggeriti in [PWN14] per avere una simulazione di riferimento (visibile nella sezione [5.1.1]), a partire dalla quale vengono poi modificati i parametri uno a uno per studiarne l'effetto.

Nel caso dei modelli stabili (nei nostri 40s di esperimento), aumentando il ritardo di comunicazione tra i veicoli, si osserva generalmente un aumento delle oscillazioni nelle velocità e nelle distanze tra i veicoli nella carovana. Questo è dovuto al fatto che un ritardo maggiore comporta una risposta più lenta del sistema di controllo, causando un accumulo di errori e una minore capacità di mantenere la stabilità. Nei grafici, questo fatto si traduce in curve più irregolari e fluttuazioni più pronunciate nelle velocità e nelle distanze tra i veicoli, specialmente in risposta ai cambiamenti repentini di velocità del primo veicolo. Si possono osservare i grafici con ritardo di comunicazione a 0.1s (in figura 23 e 24) accanto a quelli con ritardo a 0.7s (in figura 27 e 28) dove le variazioni di distanza tra i vari veicoli sono molto più accentuate (avendo ritardo di comunicazione maggiore).

D'altra parte, aumentando il *time headway*, cioè il tempo minimo di separazione tra i veicoli nella carovana, si tende ad avere un effetto opposto. Un *time headway* più lungo permette ai veicoli di avere più spazio di frenata e di reazione, riducendo il rischio di collisioni (dato che lo spazio di errore per ogni veicolo aumenta). Nei grafici troviamo infatti curve più regolari e meno oscillanti, questo si può osservare chiaramente comparando le figure con *time headway* a 0.1 (in figura 31 e 32) con quelli a 1 (in figura 33 e 34) dove le variazioni di distanza e velocità dei vari veicoli sono più sfasate dato il maggior tempo di separazione.

Possiamo anche osservare che la modifica di τ risulta in un'amplificazione o attenuazione della propagazione della velocità nel modello tra il primo ed il secondo veicolo. Questo è evidente comparando i grafici con τ a 0.01 (in figura 37 e 38) con quelli a 0.7 (in figura 39 e 40); vediamo chiaramente che la variazione di distanza tra il primo ed il secondo veicolo aumenta con τ .

Variare K_p e K_d risulta in variazioni non osservabili direttamente, soprattutto nel caso di K_d . Per K_p invece possiamo evidenziare che la sua variazione risulta in una distorsione dei grafici delle distanze e velocità. Questo fatto è visibile comparando i grafici con K_p a 0.01 (in figura 41 e 42) con quelli a 1.5 (in figura 43 e 44). Per K_d le variazioni derivative sono talmente piccole da non risultare in cambiamenti a bassa velocità.

Il numero di veicoli non influisce sulla propagazione degli errori (cioè sulla stabilità delle stringhe, come spiegato nella sezione [2.3]), infatti possiamo osservare i grafici di distanza e velocità con 10 auto (in figura 17 e 18) e non si notano differenze sulla stabilità comparandoli con i grafici di default.

Variando lo spazio tra le auto, cioè la distanza obiettivo, da $2m$ (in figura 19 e 20) a $20m$ (in figura 21 e 22) non si riscontra nessuna differenza nella dinamica della carovana, dato che nei nostri esperimenti partiamo sempre da $1m$ in più di distanza tra ogni veicolo rispetto a quella obiettivo. La variazione della distanza obiettivo però può influenzare in alcuni casi la stabilità del modello, lasciando più o meno spazio agli errori di distanza.

Cambiando la velocità del primo veicolo in un grafico costante, come si può vedere nelle figure 49, 50 e 51, 52, risulta che le velocità di tutti gli altri veicoli del *platoon* tende al valore del primo, mentre mantengono una distanza dipendente dal *time headway* (identificabile come la distanza di sicurezza richiesta dal modello).

Possiamo vedere che lasciando i parametri di default e modificando il grafico della velocità del primo veicolo con punti 20-0-20-0-20 m/s (in figura 53 e 54) oppure 35-35-0-35-35 m/s (in figura 55 e 56) il modello performa ottimamente, adattandosi ai cambiamenti della vettura pilota, senza essere particolarmente influenzato dal ritardo di comunicazione di $0.2s$. Soprattutto nel caso del secondo esperimento possiamo vedere che la stabilità è mantenuta anche considerando la variazione di velocità massima, cioè da $35m/s$ a $0m/s$.

Infatti, la maggior parte delle configurazioni dei parametri risultano in modelli stabili se consideriamo variazioni di velocità del veicolo pilota non eccessive (equiparabili a quelle possibili in una situazione realistica). Abbiamo quindi etichettato come modelli instabili anche quelli che nei $40s$ di esperimento sono vicini ad una collisione, cioè nel raggio di circa $1m$ dalla vettura precedente. L'instabilità di questi esperimenti è stata comunque confermata osservandoli tramite l'interfaccia grafica per una durata più lunga.

I modelli risultati instabili, cioè che sono falliti o andati vicino al fallimento nel tempo dei nostri esperimenti, sono riassunti nella sezione [5.2].

5.1.1 Parametri di default

Questi parametri sono presi dall'articolo [PWN14].

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.7

Velocità t_1	Velocità t_2	Velocità t_3	Velocità t_4	Velocità t_5
2 m/s	4 m/s	6 m/s	8 m/s	10 m/s

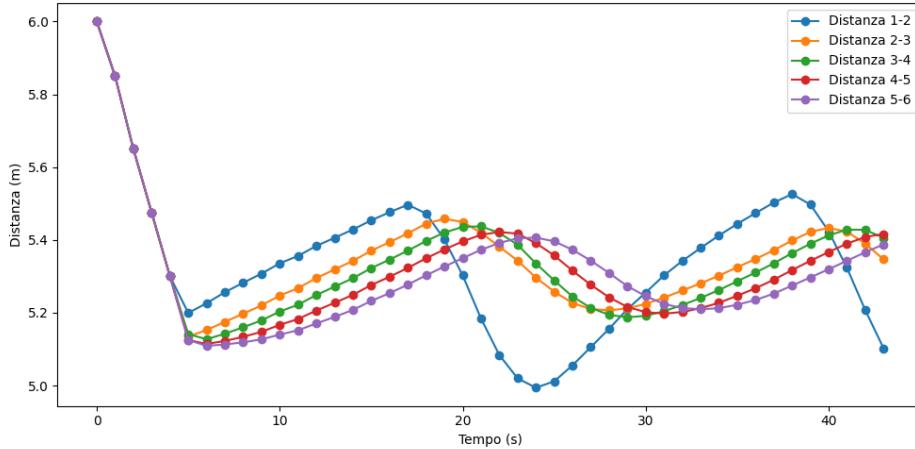


Figura 13: Distanze con parametri di default.

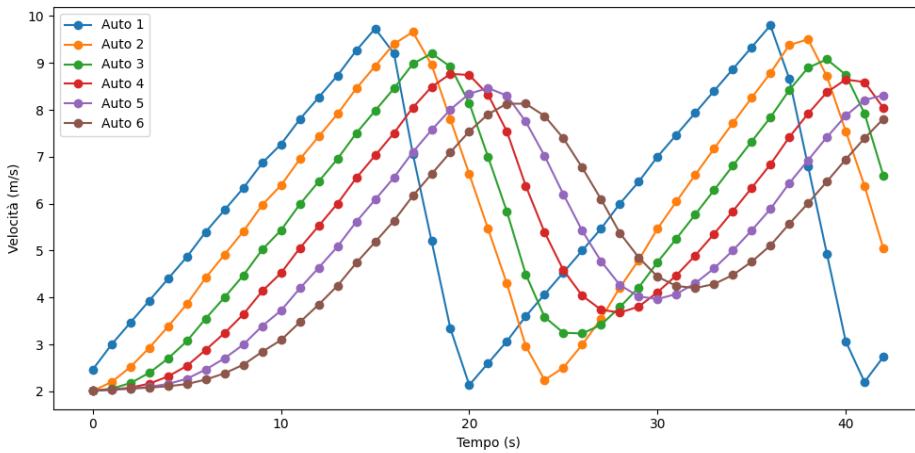


Figura 14: Velocità con parametri di default.

5.1.2 Numero di auto (m)

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
8	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.7

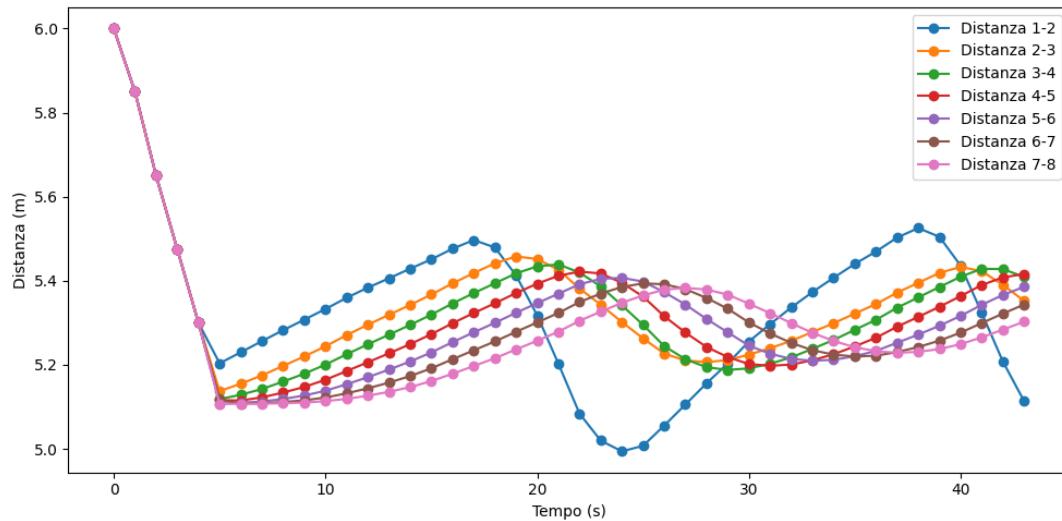


Figura 15: Distanze con 8 auto.

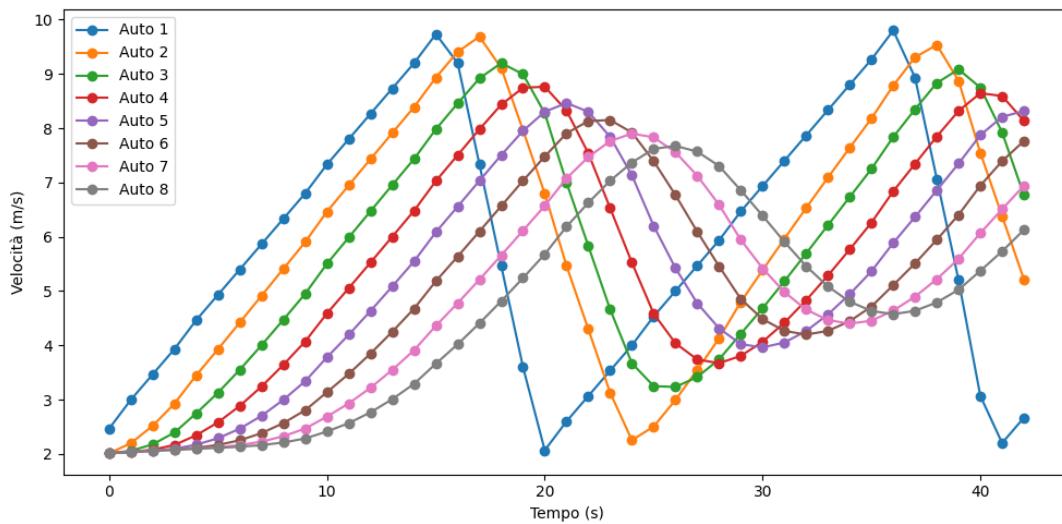


Figura 16: Velocità con 8 auto.

Esperimenti

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
10	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.7

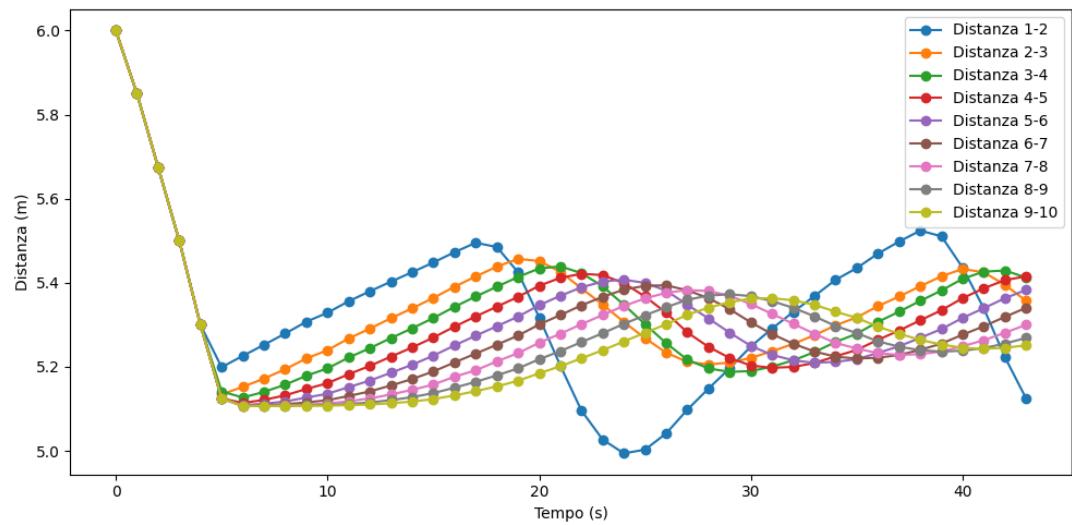
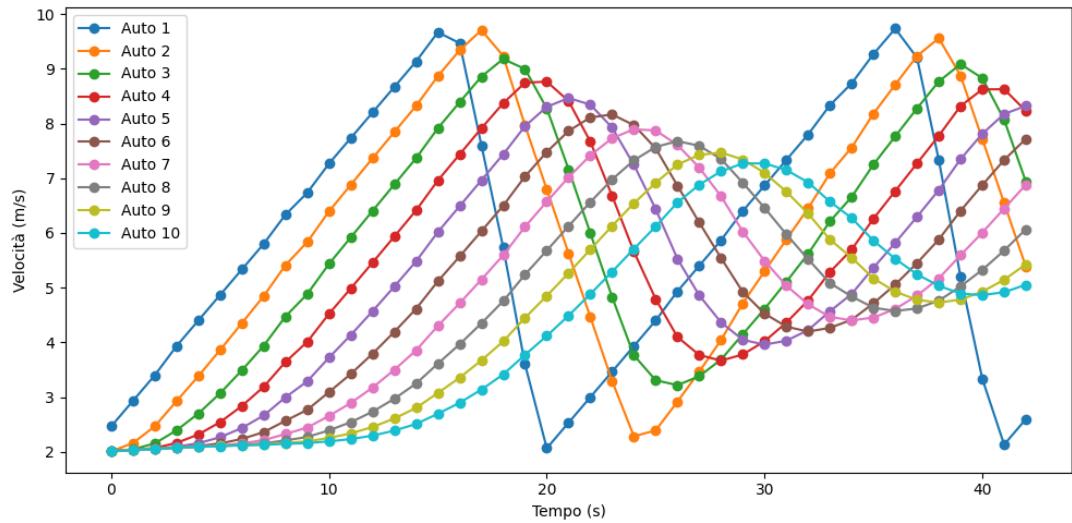
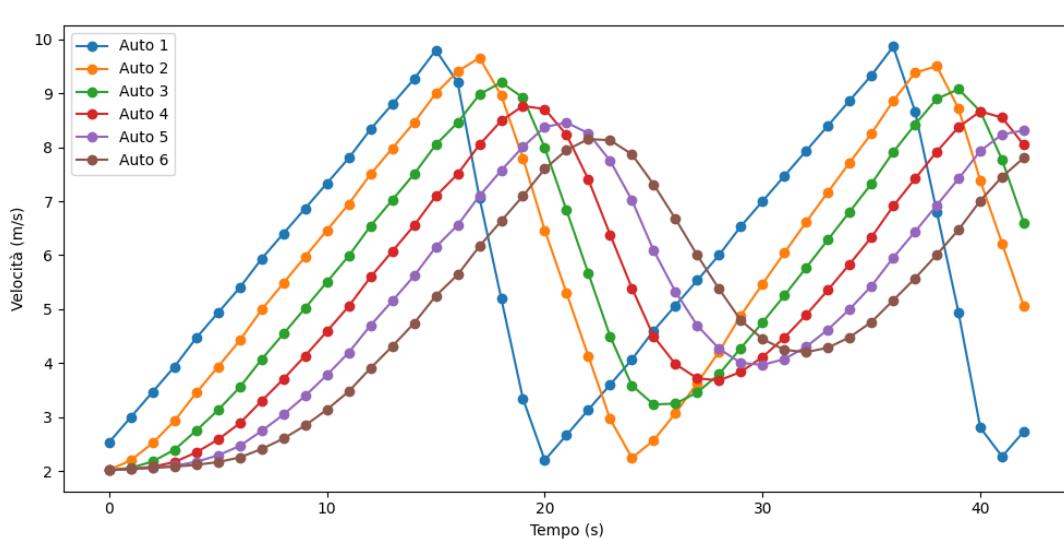
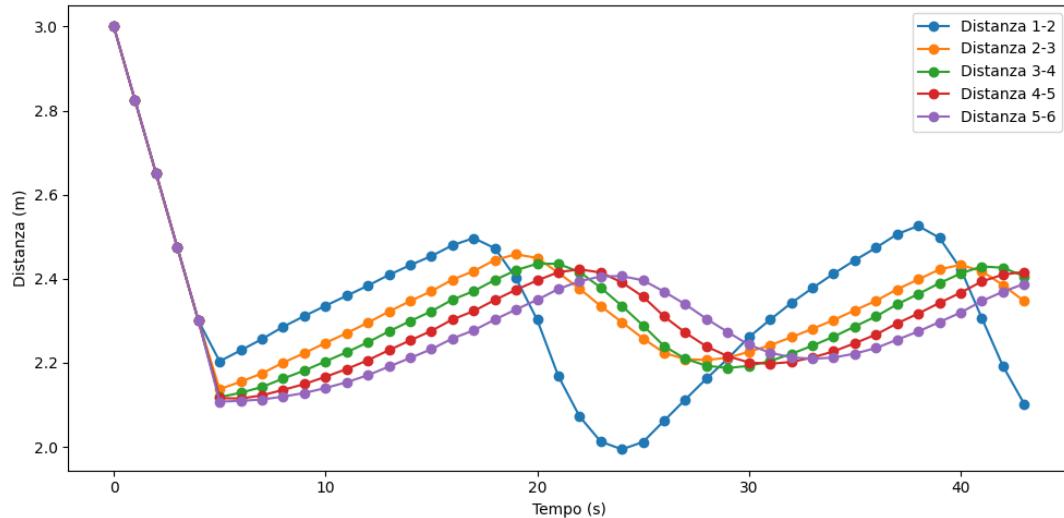


Figura 17: Distanze con 10 auto.



5.1.3 Spazio tra le auto (d_i)

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	3.0m	2.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.7



Esperimenti

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	21.0m	20.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.7

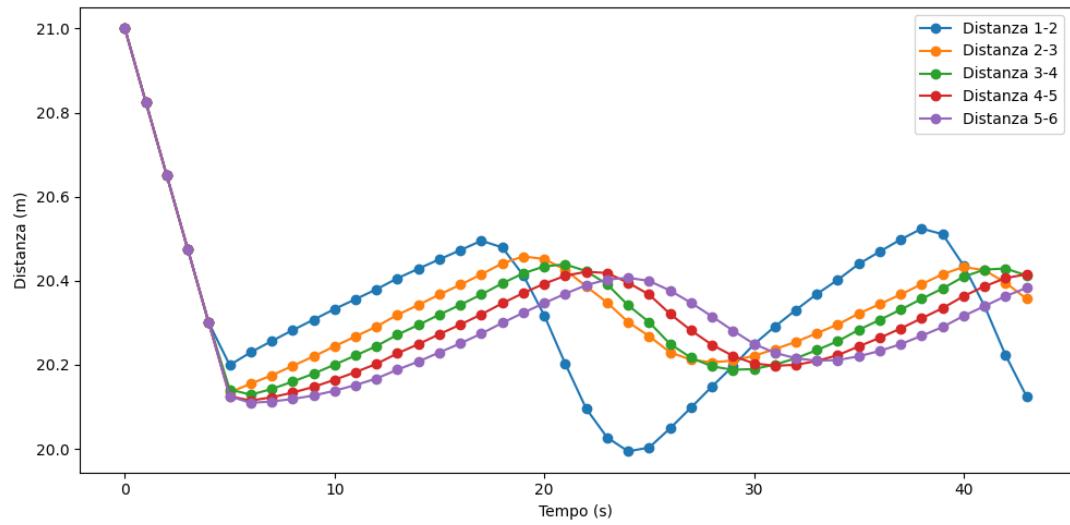
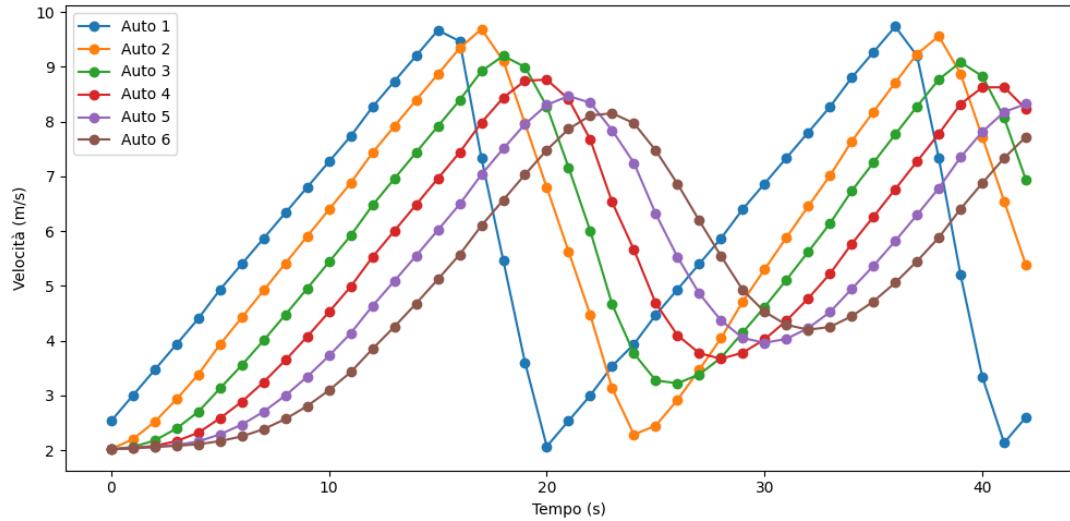


Figura 21: Distanze con 20.0m come distanza obiettivo.



5.1.4 Ritardo di comunicazione

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.1s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.7

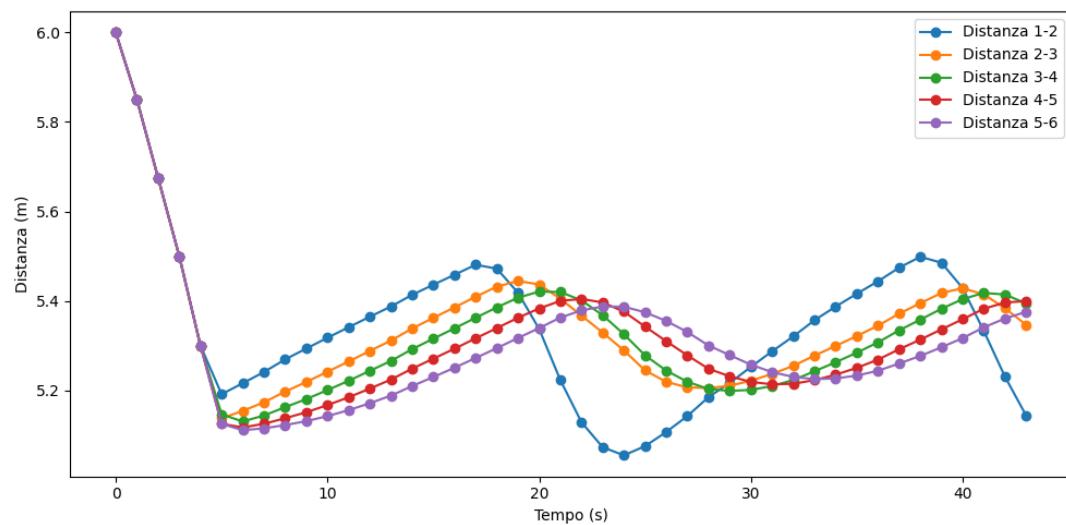


Figura 23: Distanze con ritardo a 0.1s.

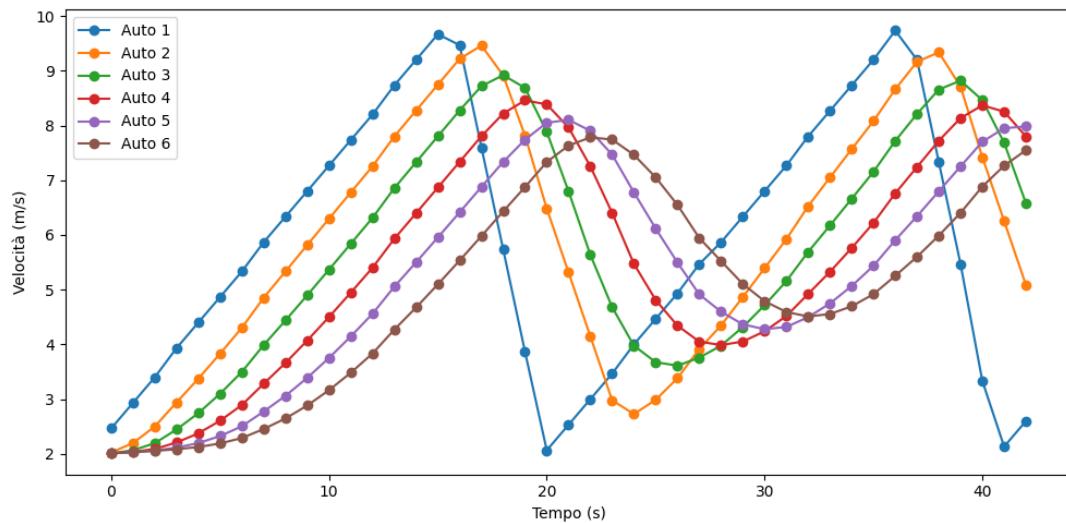


Figura 24: Velocità con ritardo a 0.1s.

Esperimenti

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.5s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.7

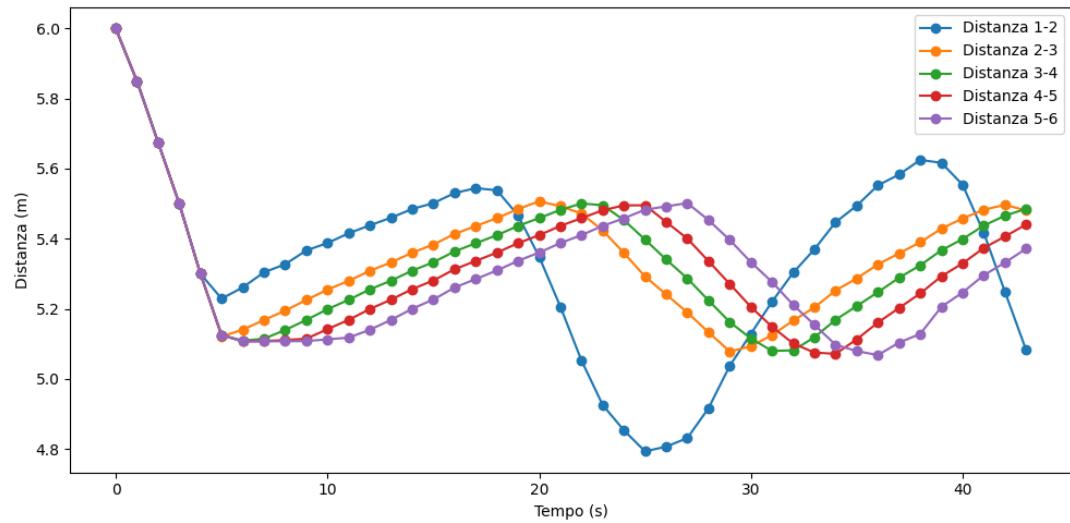


Figura 25: Distanze con ritardo a 0.5s.

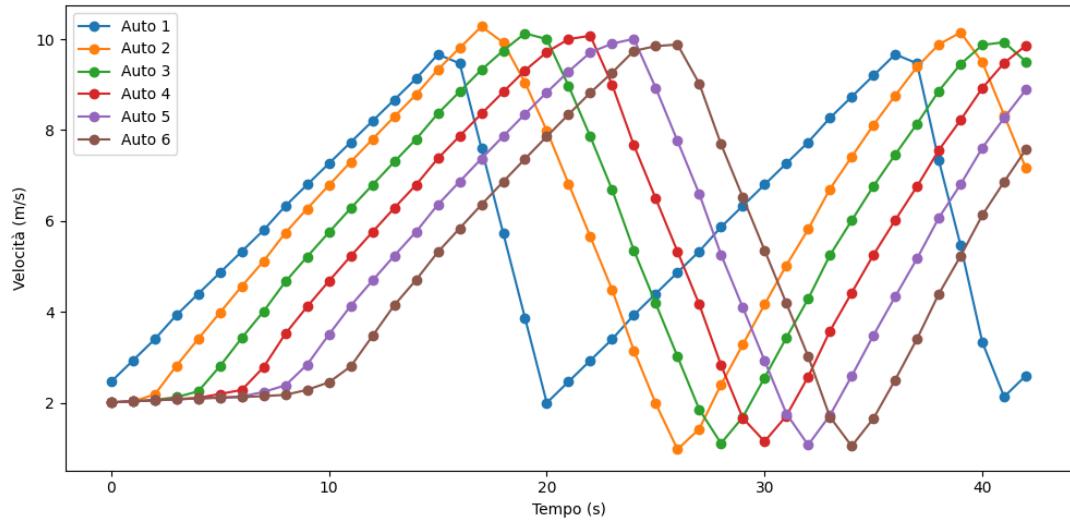


Figura 26: Velocità con ritardo a 0.5s.

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.7s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.7

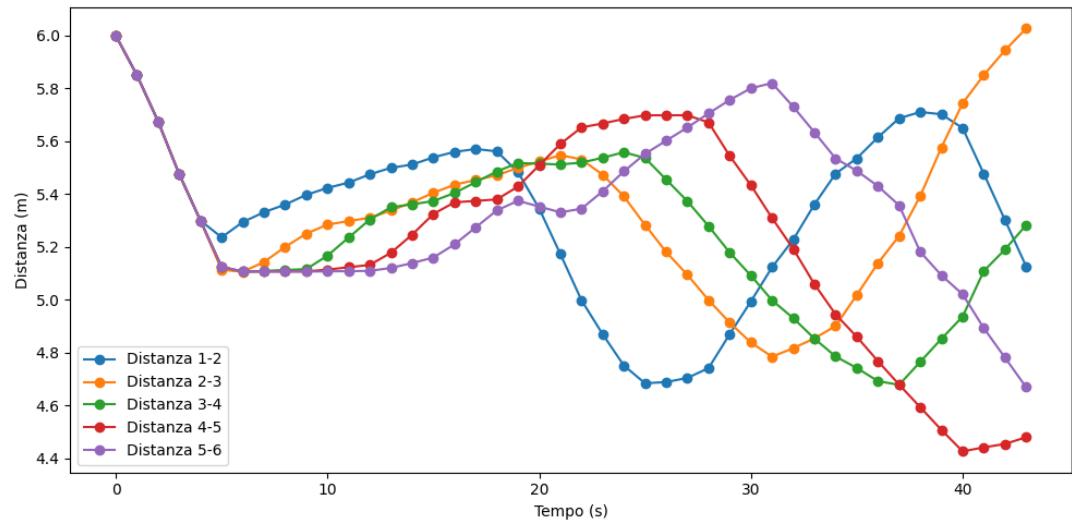


Figura 27: Distanze con ritardo a 0.7s.

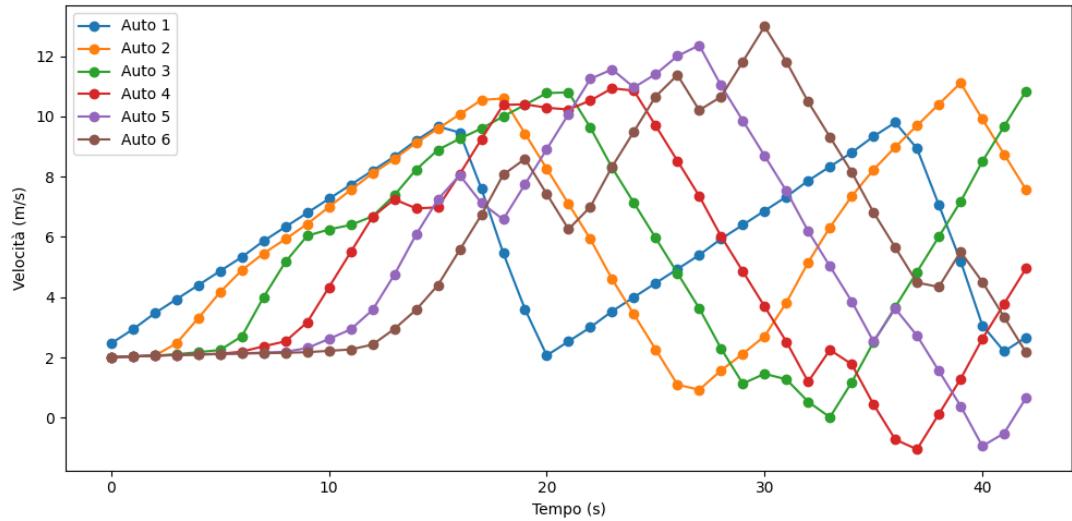


Figura 28: Velocità con ritardo a 0.7s.

Esperimenti

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	1s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.7

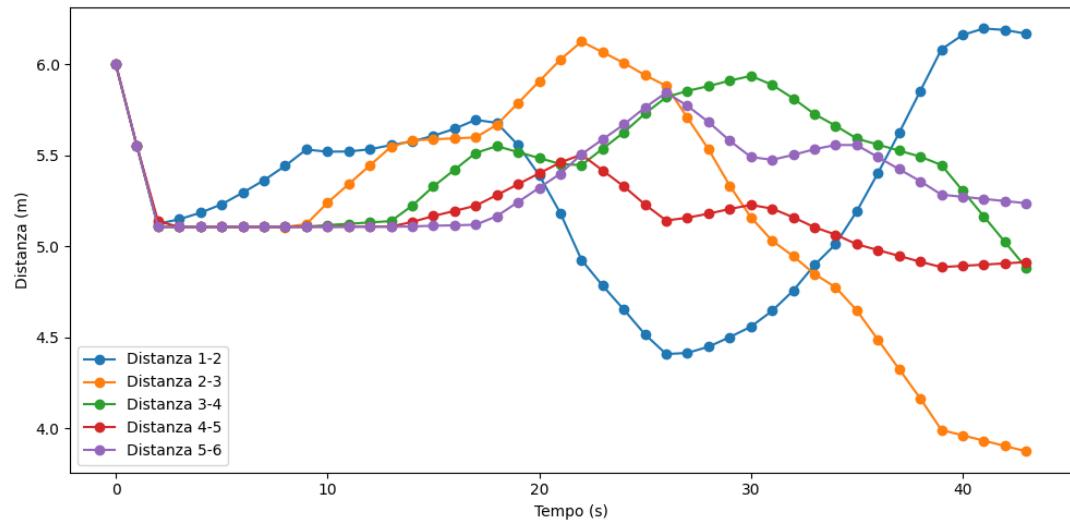


Figura 29: Distanze con ritardo a 1s.

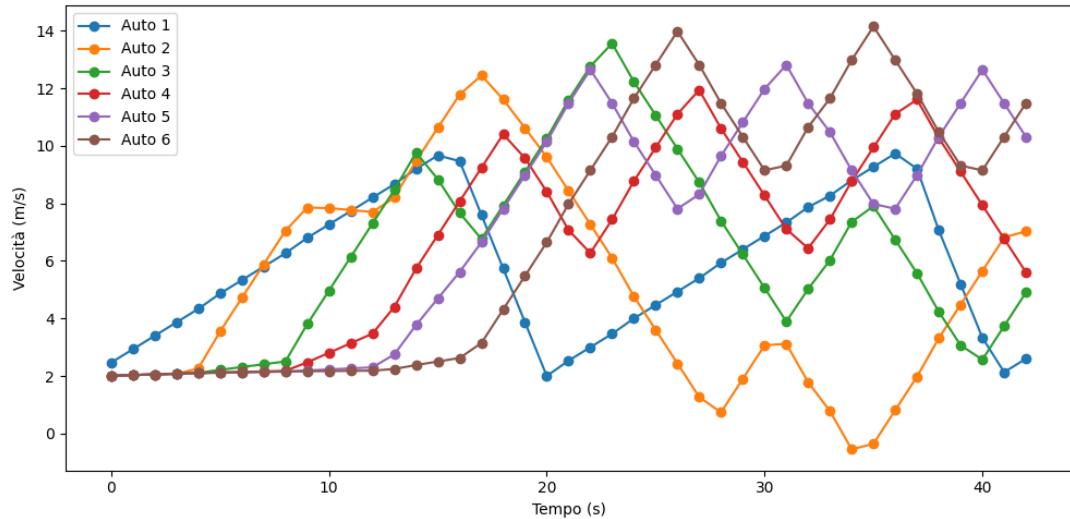


Figura 30: Velocità con ritardo a 1s.

5.1.5 Time Headway (h)

La descrizione teorica di questo parametro si trova nella sezione [2.2.1].

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.1s	0.1	0.2	0.7

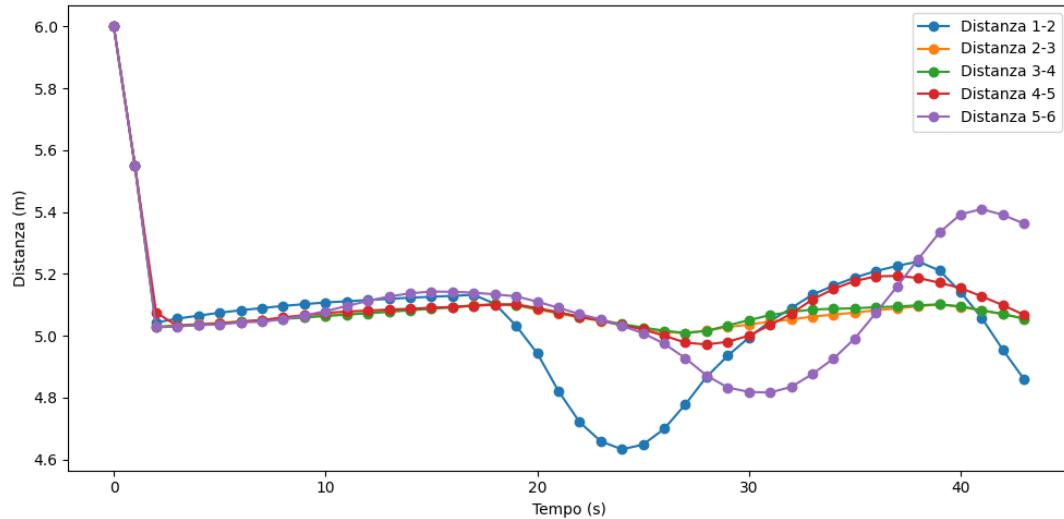


Figura 31: Distanze con *time headway* a 0.1s.

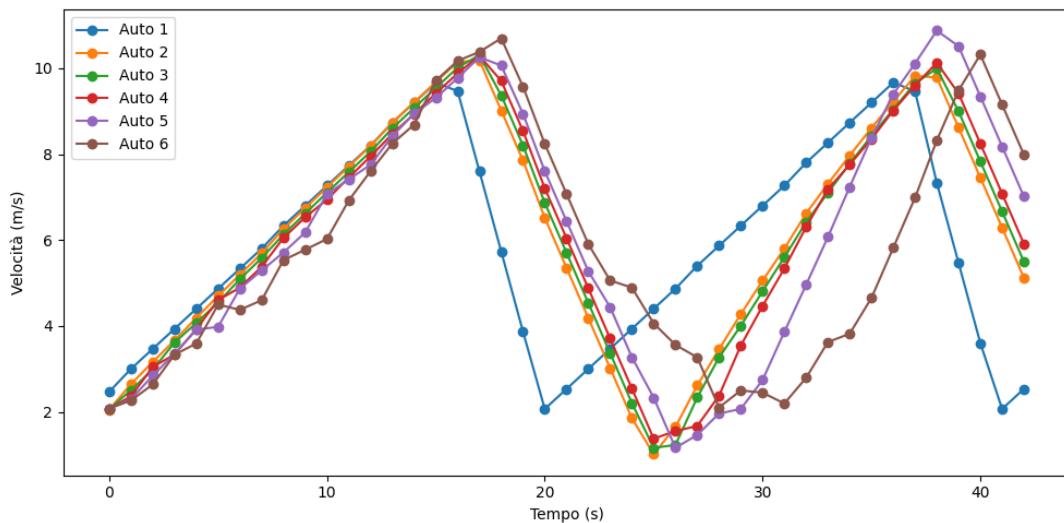


Figura 32: Velocità con *time headway* a 0.1s.

Esperimenti

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
1s	0.1	0.2	0.7

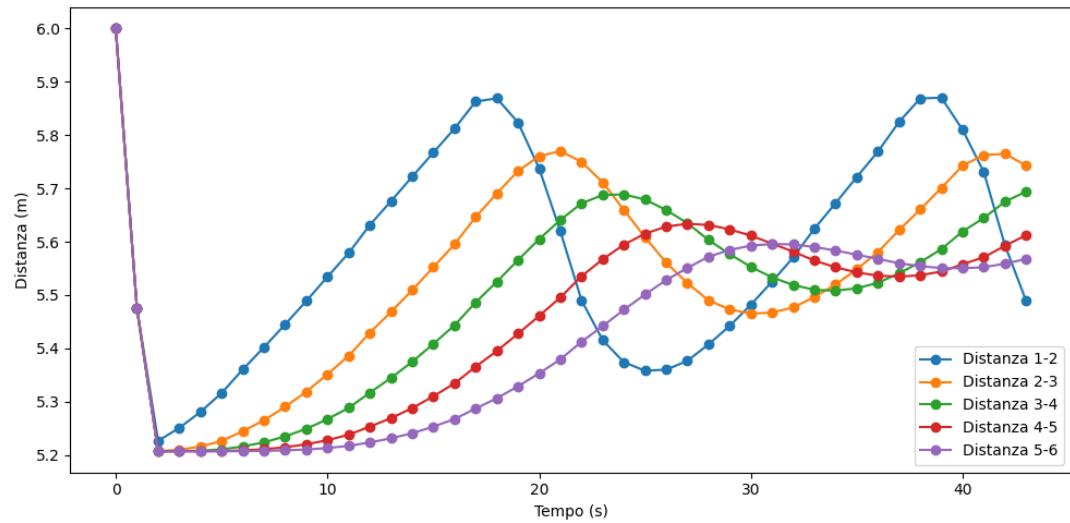


Figura 33: Distanze con *time headway* a 1s.

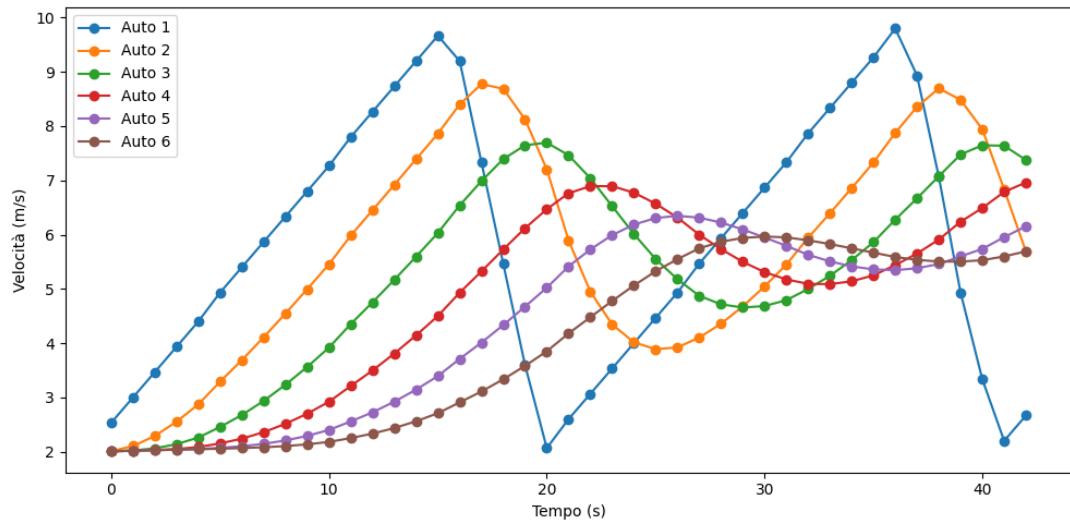
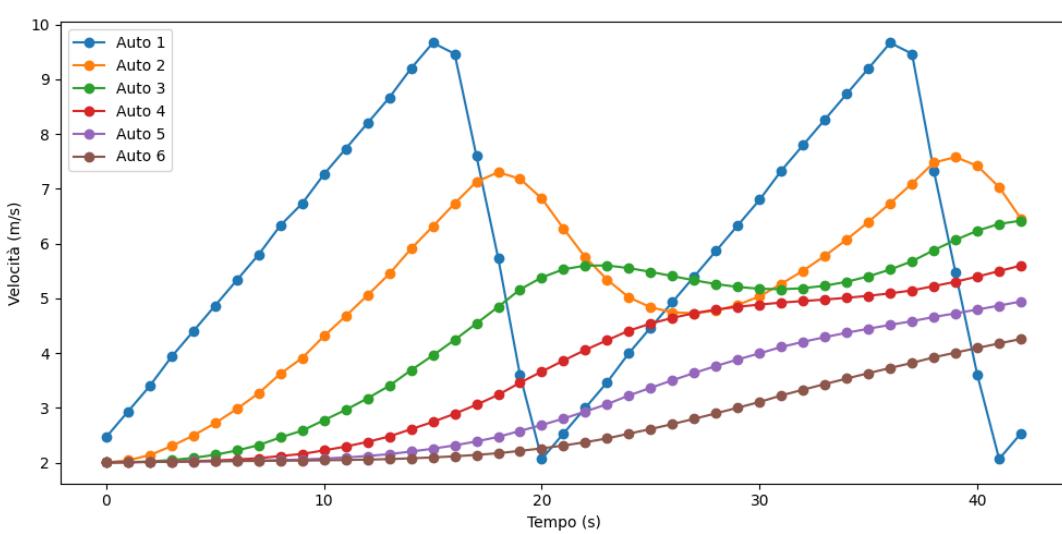
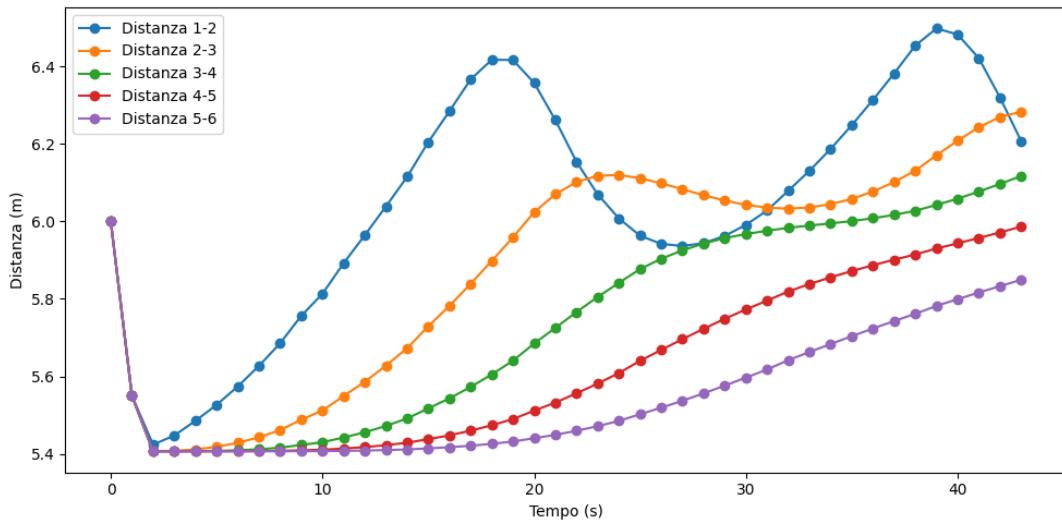


Figura 34: Velocità con *time headway* a 1s.

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
2s	0.1	0.2	0.7



5.1.6 Tau (τ)

La descrizione teorica di questo parametro si trova nella sezione [2.2.2].

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.01	0.2	0.7

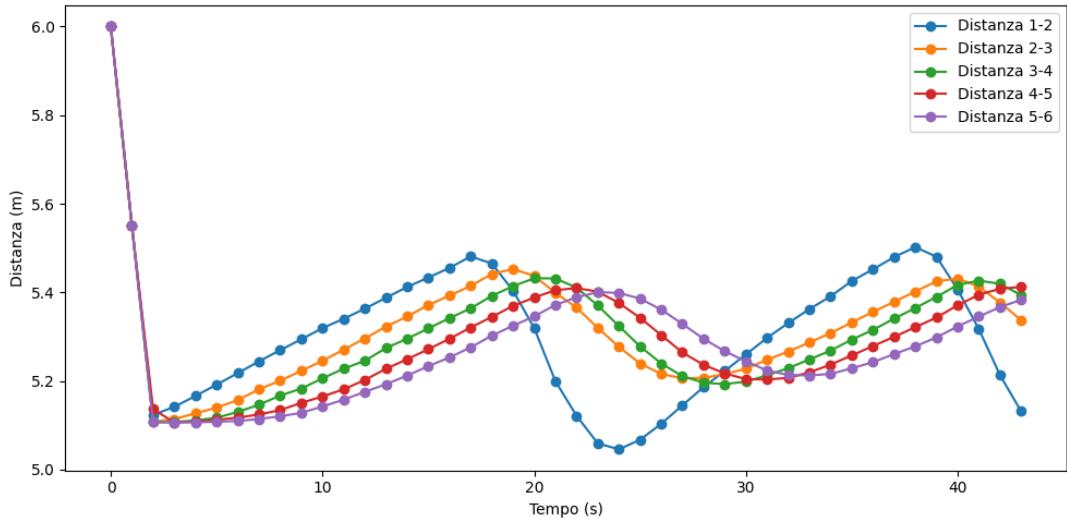


Figura 37: Distanze con τ a 0.01.

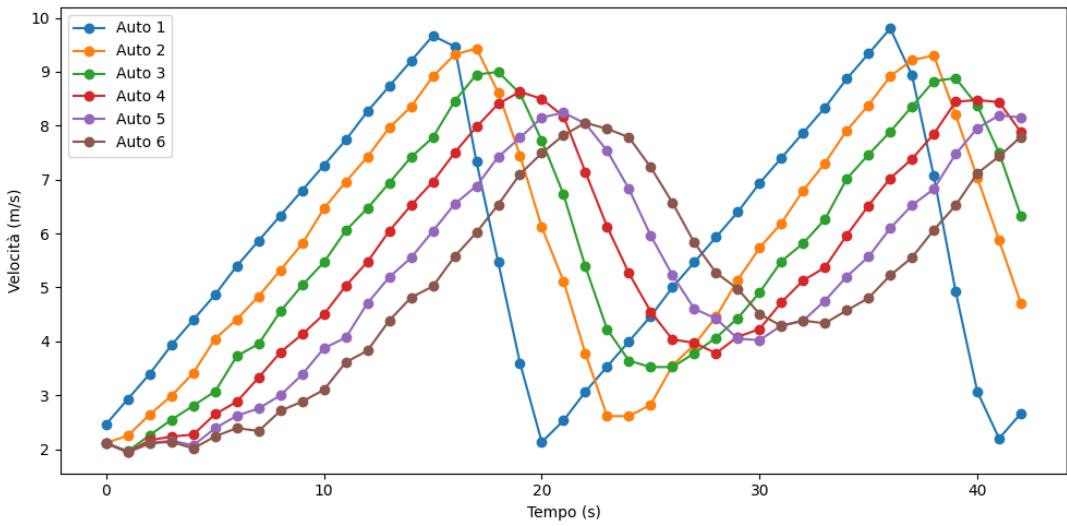
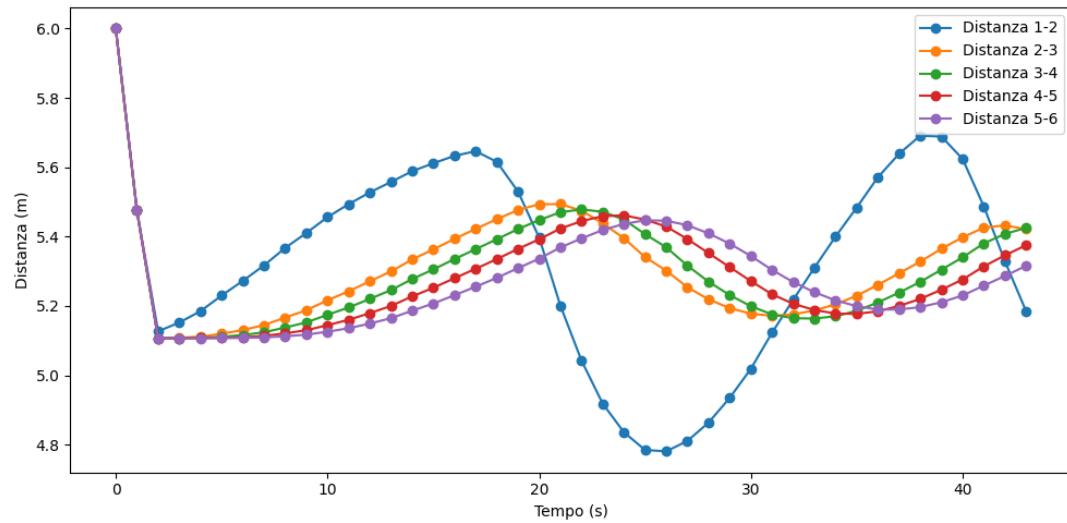
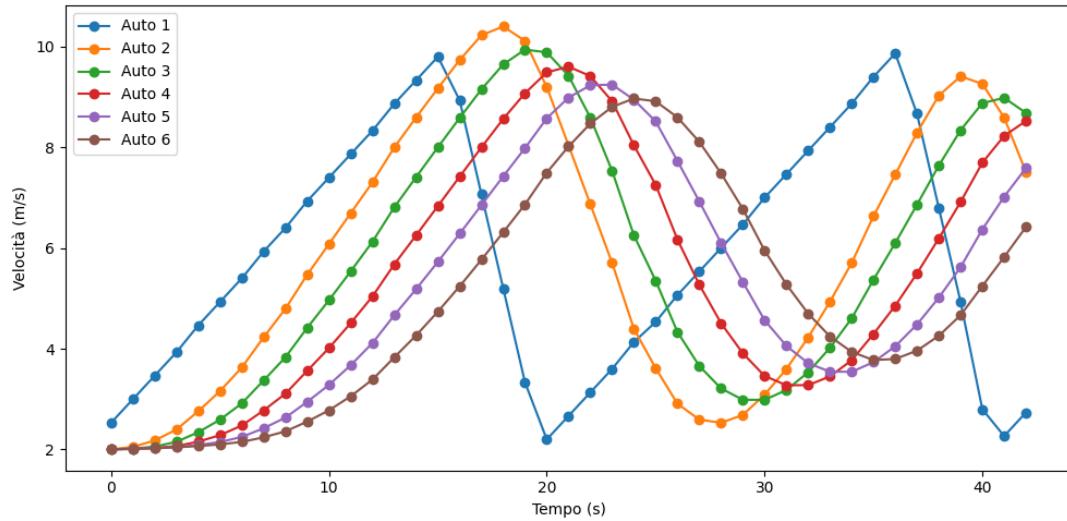


Figura 38: Velocità con τ a 0.01.

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.7	0.2	0.7

Figura 39: Distanze con τ a 0.7.Figura 40: Velocità con τ a 0.7.

5.1.7 Kp (k_p)

La descrizione teorica di questo parametro si trova nella sezione [2.2.3].

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.01	0.7

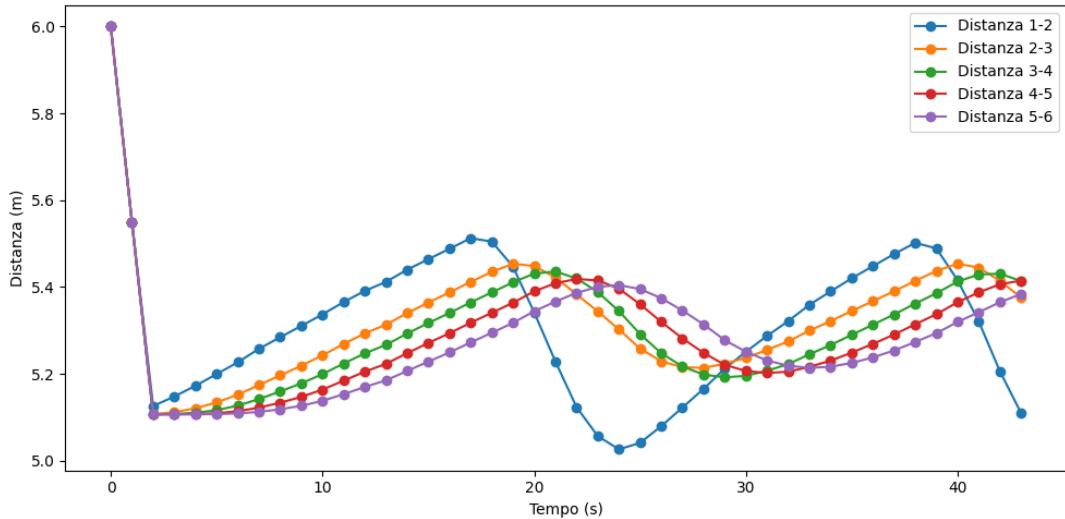


Figura 41: Distanze con k_p a 0.01.

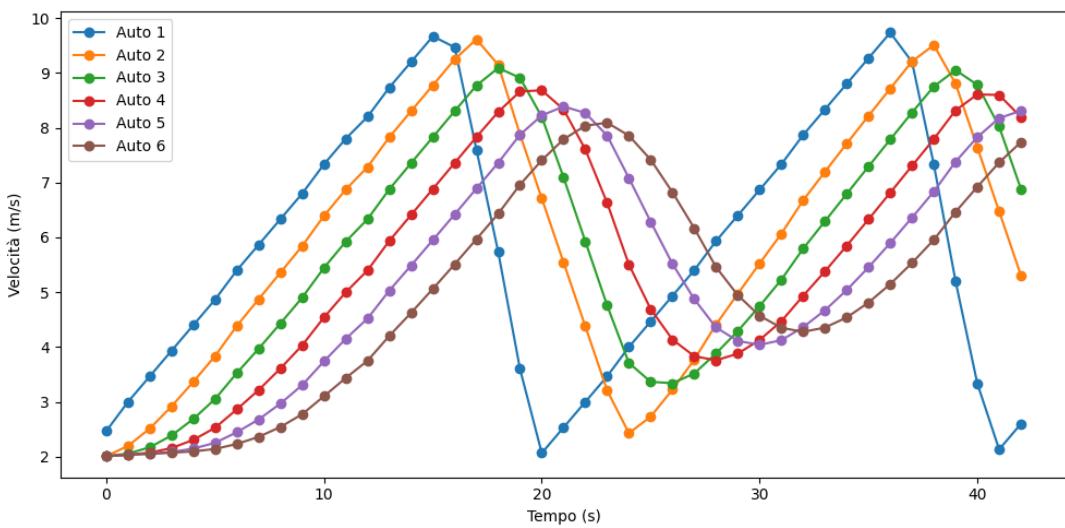
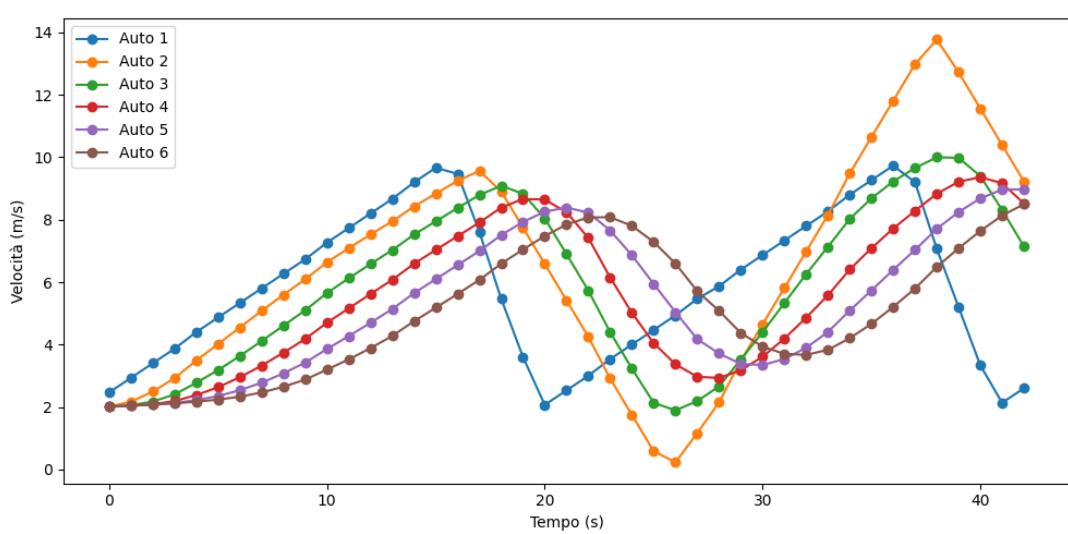
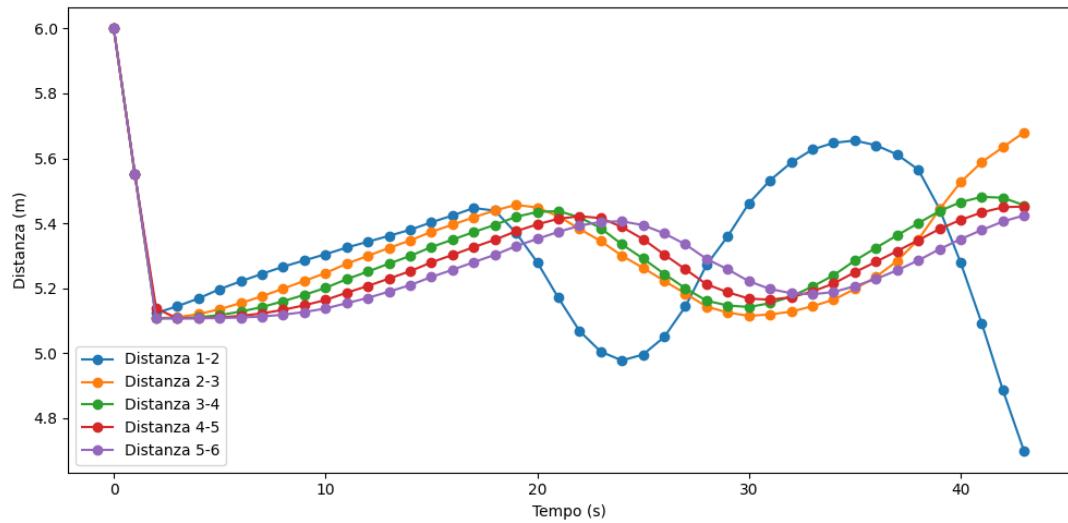


Figura 42: Velocità con k_p a 0.01.

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	1.5	0.7



5.1.8 Kd (k_d)

La descrizione teorica di questo parametro si trova nella sezione [2.2.3].

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.1

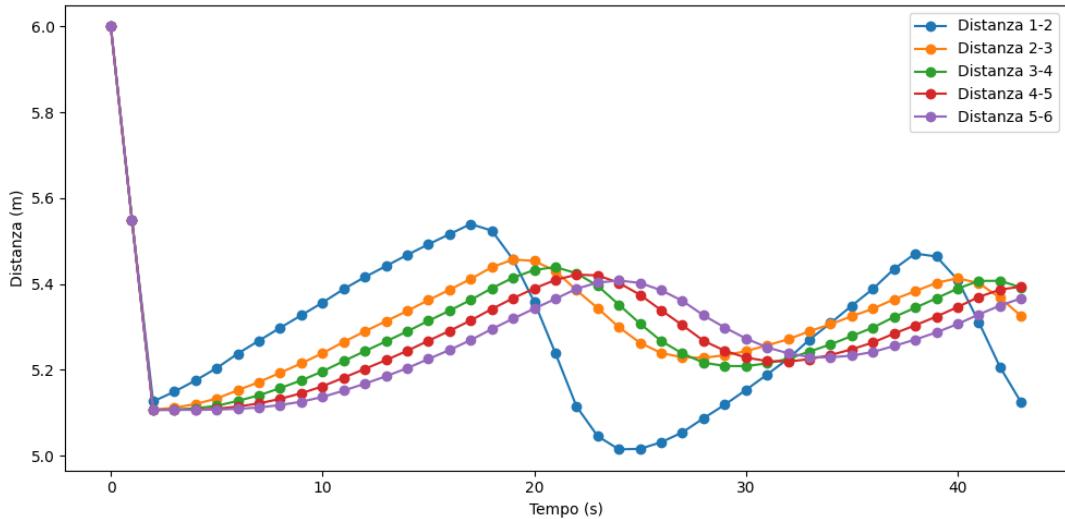


Figura 45: Distanze con k_d a 0.1.

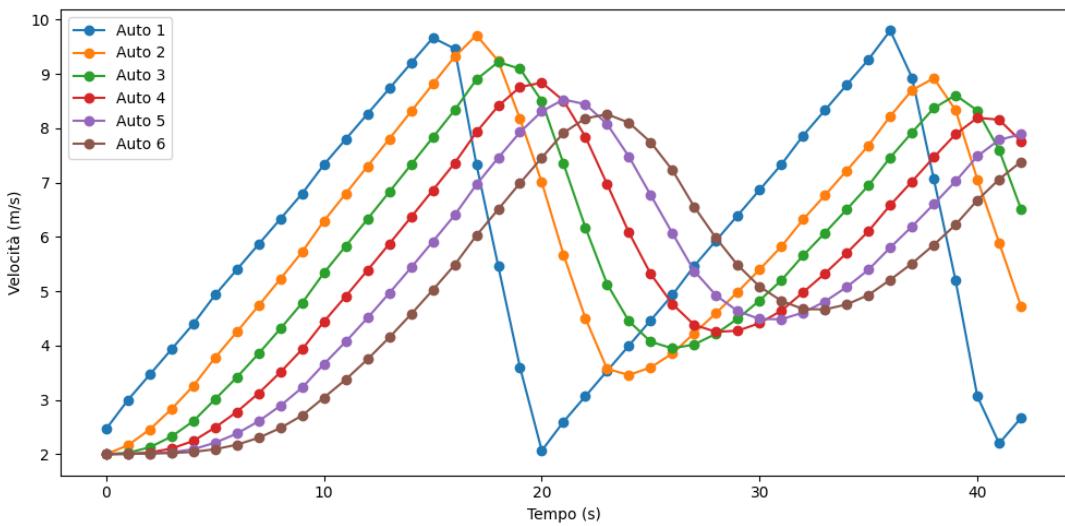
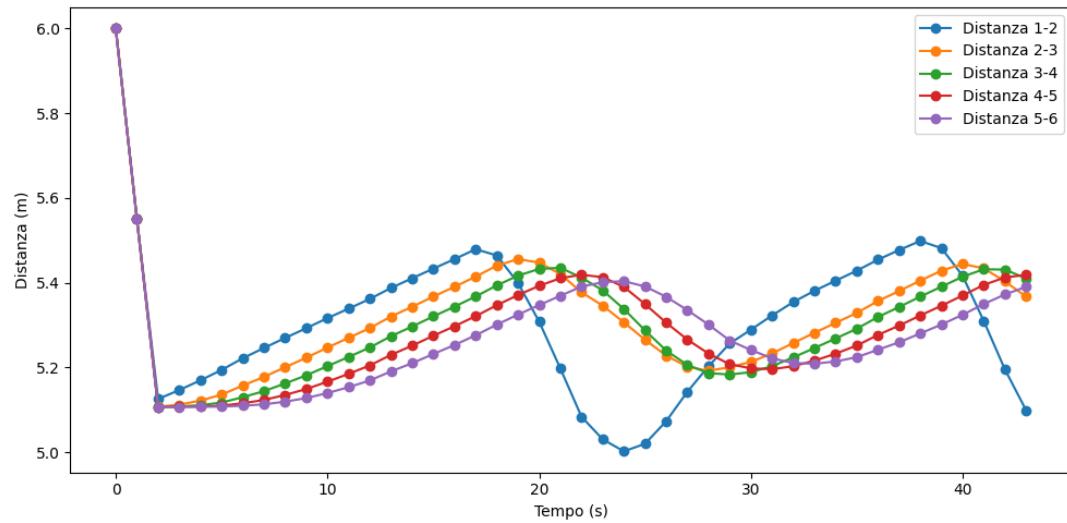
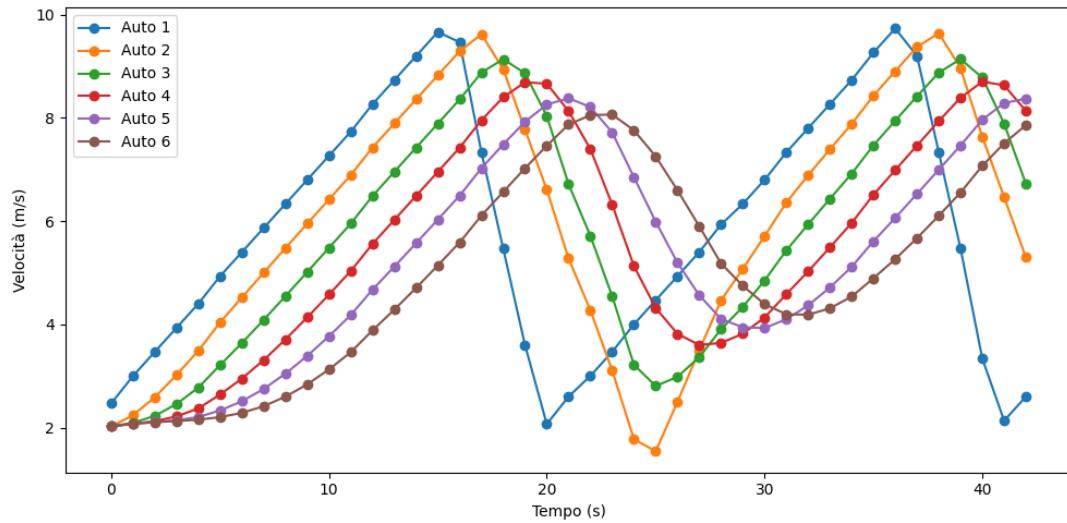


Figura 46: Velocità con k_d a 0.1.

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	1.5

Figura 47: Distanze con k_d a 1.5.Figura 48: Velocità con k_d a 1.5.

5.1.9 Velocità (v_1) con parametri di default

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.7

Velocità t_1	Velocità t_2	Velocità t_3	Velocità t_4	Velocità t_5
10 m/s				

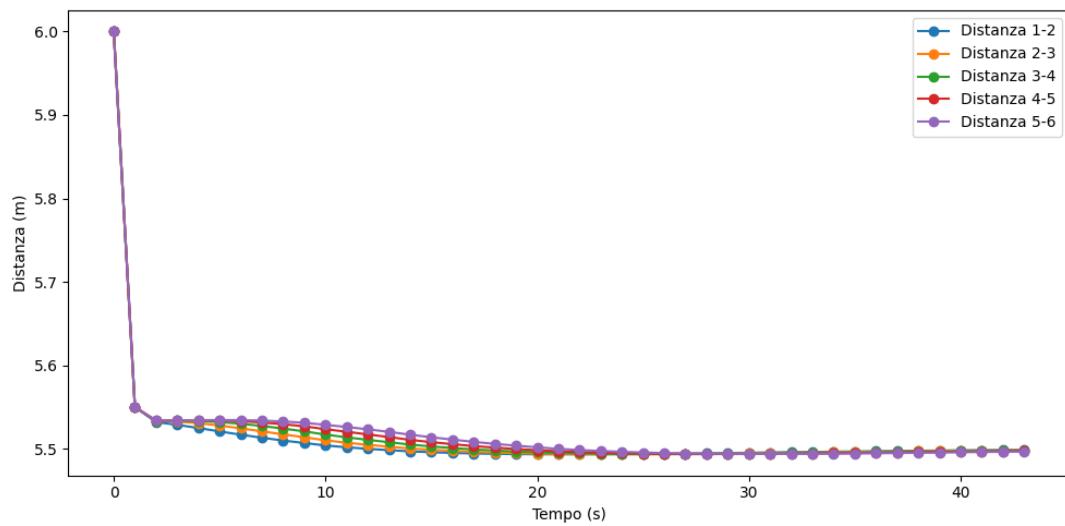


Figura 49: Distanze con velocità pilota costante a 10 m/s.

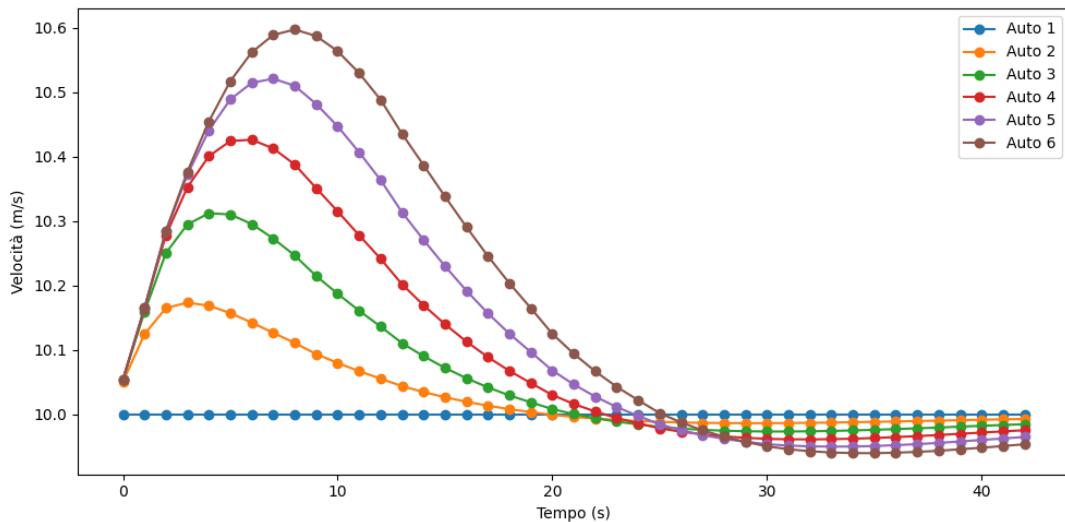


Figura 50: Velocità con velocità pilota costante a 10 m/s.

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.7

Velocità t_1	Velocità t_2	Velocità t_3	Velocità t_4	Velocità t_5
35 m/s				

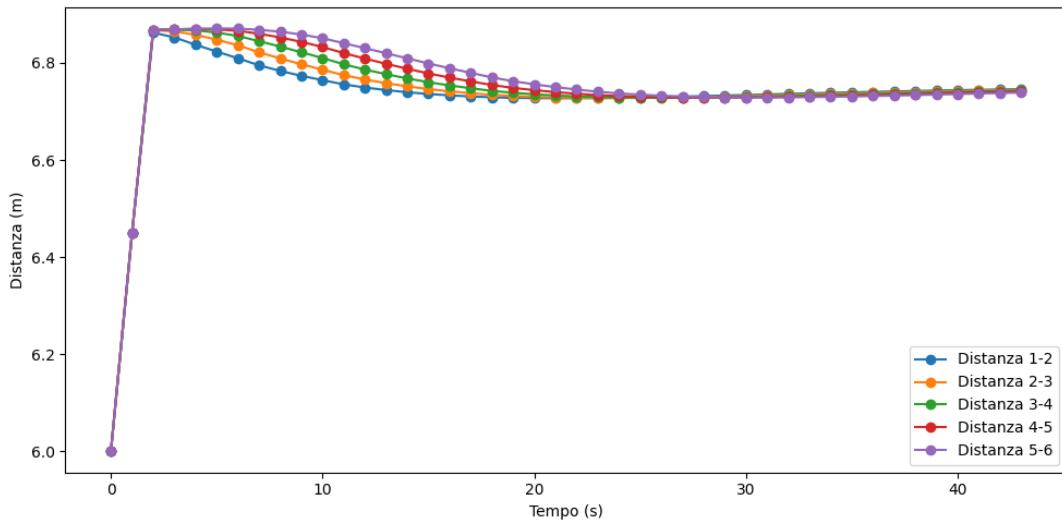


Figura 51: Distanze con velocità pilota costante a 35 m/s.

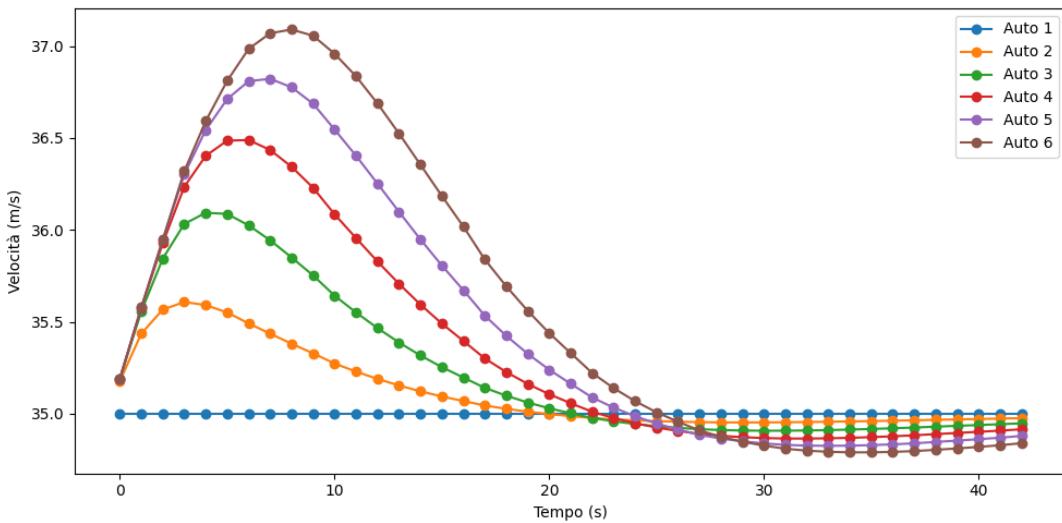


Figura 52: Velocità con velocità pilota costante a 35 m/s.

Esperimenti

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.7

Velocità t_1	Velocità t_2	Velocità t_3	Velocità t_4	Velocità t_5
20 m/s	0 m/s	20 m/s	0 m/s	20 m/s

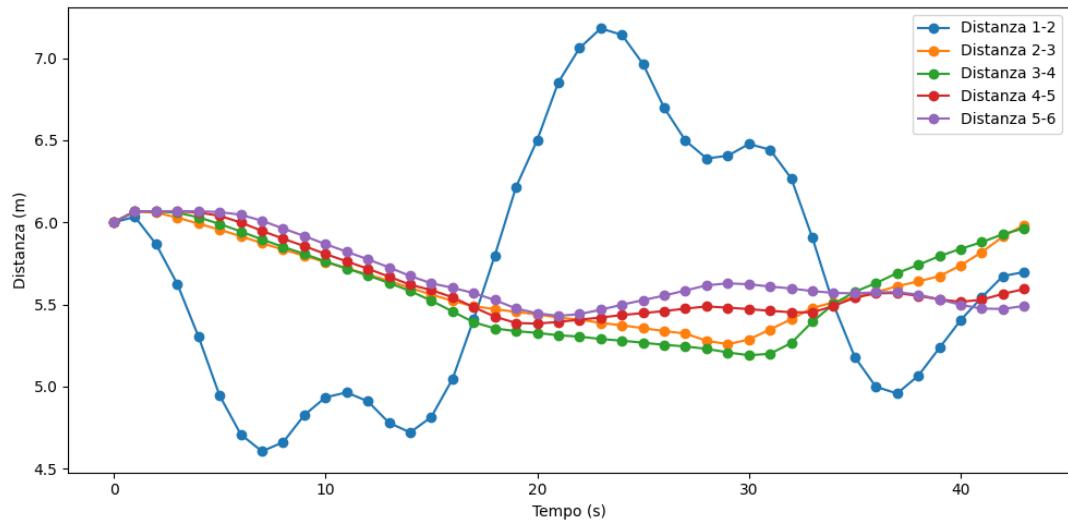
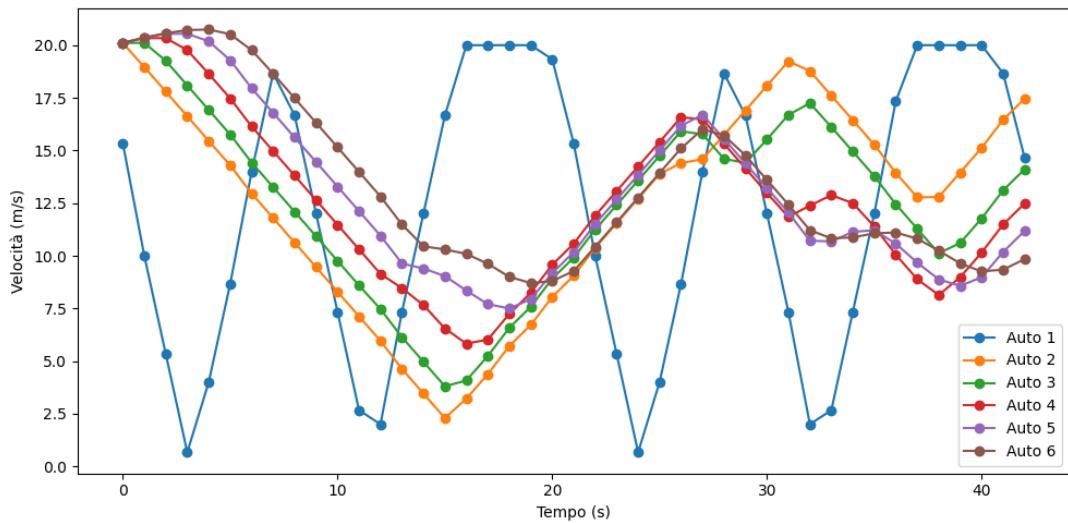
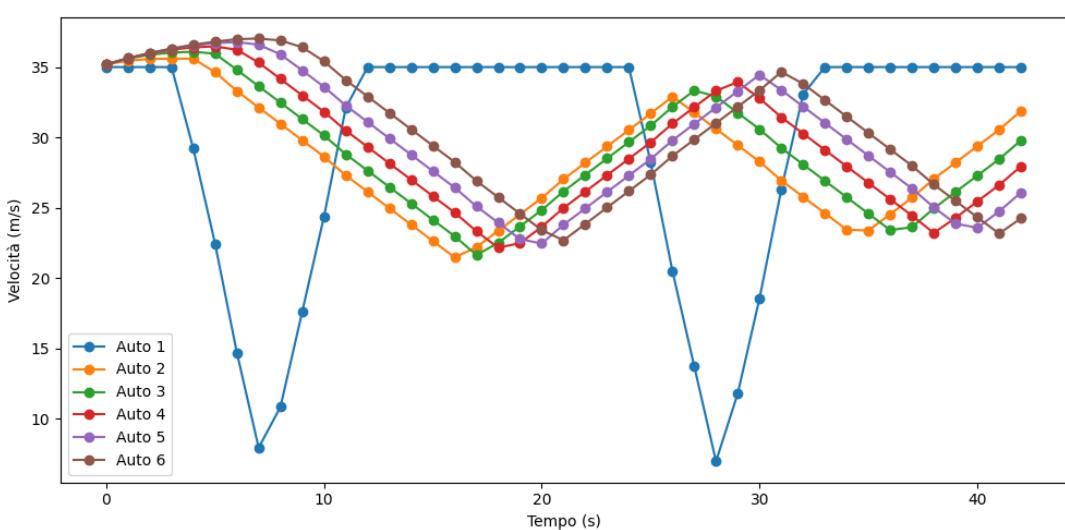
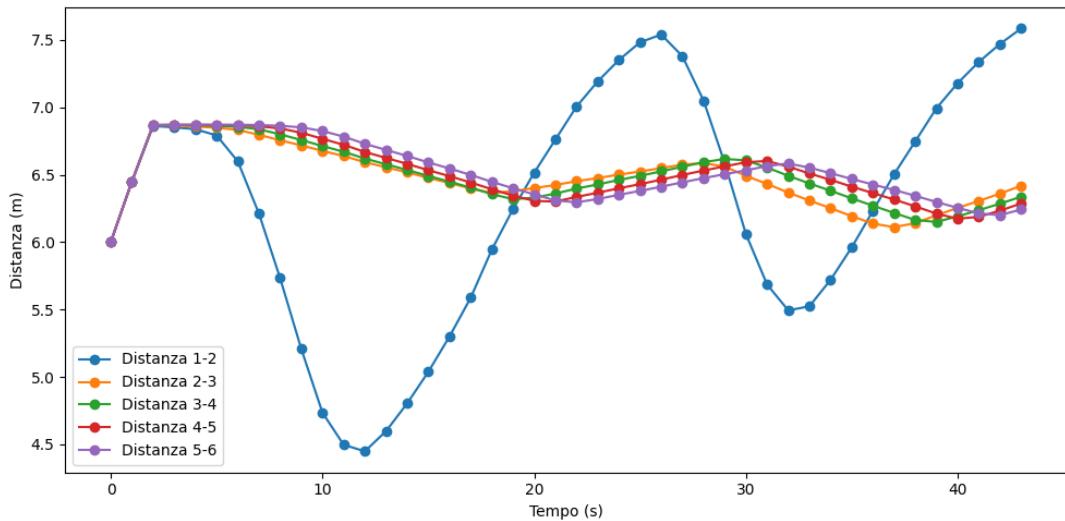


Figura 53: Distanze con velocità pilota variabile tra 20 m/s e 0 m/s.



N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.7

Velocità t_1	Velocità t_2	Velocità t_3	Velocità t_4	Velocità t_5
35 m/s	35 m/s	0 m/s	35 m/s	35 m/s



5.2 Modelli instabili

Possiamo osservare in vari esperimenti stabili che i veicoli viaggiano anche a velocità negative, questo per compensare le distanze tra di loro e tornare verso la distanza obiettivo. Nel nostro caso questi sono considerati modelli stabili ma in uno scenario reale questo comportamento potrebbe non essere desiderato.

Come abbiamo visto nella sezione precedente, i parametri che più influiscono sulla dinamica del *platoon* sono il ritardo di comunicazione ed il *time headway*. Oltre a questi anche il grafico delle velocità del veicolo pilota, se presenta variazioni abbastanza grandi, può rendere instabile il modello. Per chiarezza un modello instabile è definito da una combinazione di parametri che portano (nei 40s di esperimento) vicino al fallimento, come spiegato in precedenza.

L'aumento del ritardo di comunicazione danneggia particolarmente le prestazioni del modello; possiamo infatti osservare dai grafici delle distanze e delle velocità dei veicoli con ritardo di comunicazione a 3s (in figura 58 e 57) e a 4s (in figura 60 e 59) che le vetture successive rispondono troppo tardi alle variazioni di velocità delle vetture precedenti, portando così a correzioni di velocità errate ed infine all'instabilità.

Il grafico delle distanze in figura 65 e quello delle velocità in figura 66 sono i risultati di un esperimento composito che modifica la distanza obiettivo, la dinamica del primo veicolo ed il *time headway*. In particolare il *time headway*, diminuito rispetto al valore di default di 0.5s, non è abbastanza grande per compensare la propagazione degli errori (dovuti al cambio rapido della velocità del primo veicolo) fino alle ultime vetture del *platoon*.

Il grafico delle distanze in figura 67 e quello delle velocità in figura 68 sono invece i risultati di un esperimento composito che va ad aumentare il ritardo di comunicazione, la dinamica del primo veicolo e diminuire il *time headway* a 0.3s. Con la combinazione di parametri usata l'esperimento fallisce in soli 11s, rispetto ai 40s nominali, dato che ha *time headway* troppo basso e ritardo di comunicazione troppo alto considerando la distanza obiettivo di soli 3.5m.

In conclusione il modello descritto in [PWN14] non soffre particolarmente le variazioni dei suoi parametri di controllo k_p , k_d e τ ma viene influenzato in maggior parte da due parametri: il *time headway* ed il ritardo di comunicazione. Questi sono infatti parametri che impongono al modello dei vincoli stringenti, da una parte viene modificato il tempo di reazione disponibile ad un certo veicolo e dall'altra viene aggiunto un ritardo alla propagazione delle informazioni all'interno del *platoon*.

5.2.1 Ritardo di comunicazione

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	3s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.7

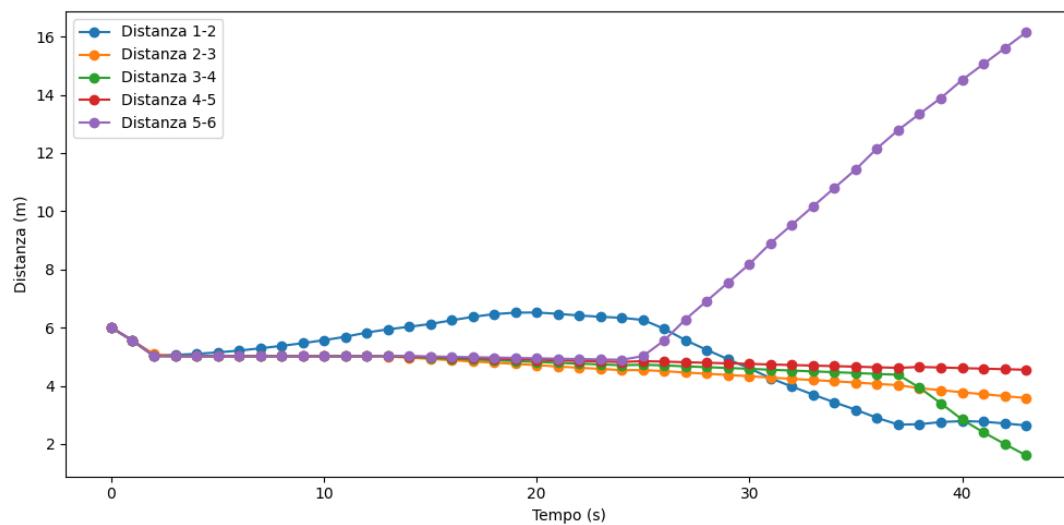


Figura 57: Distanze con ritardo a 3s.

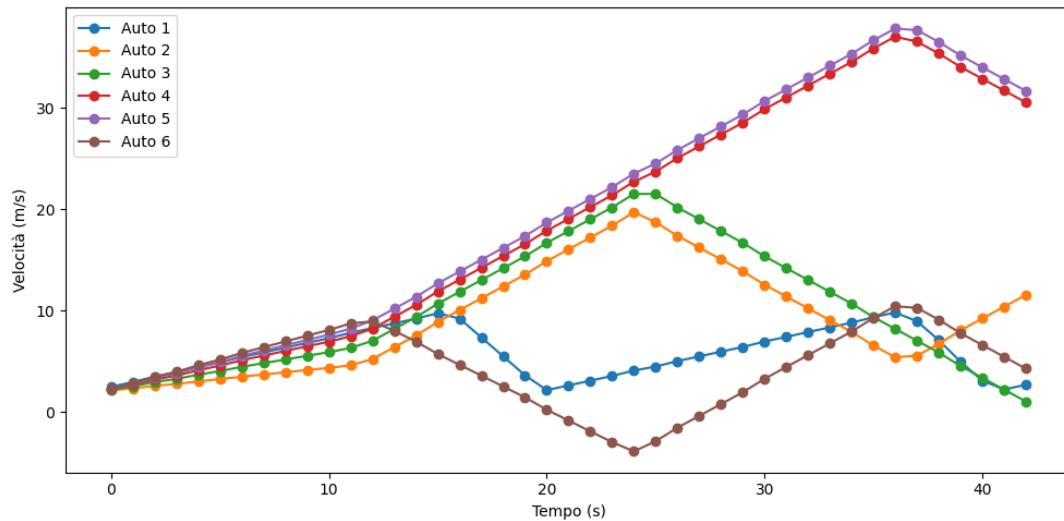


Figura 58: Velocità con ritardo a 3s.

Esperimenti

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	6.0m	5.0m	4s
Time Headway (h)	Tau (τ)	k_p	k_d
0.5s	0.1	0.2	0.7

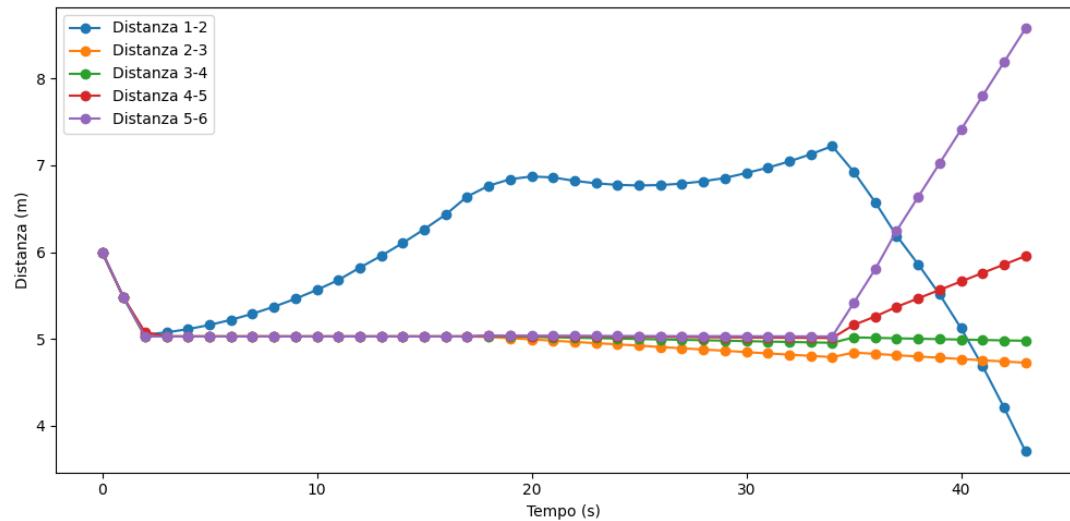


Figura 59: Distanze con ritardo a 4s.

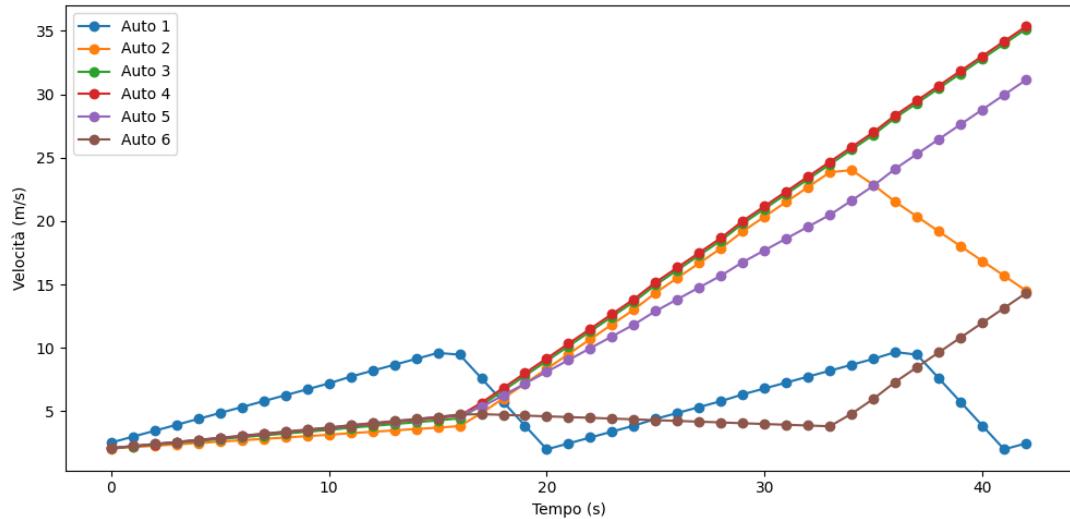


Figura 60: Velocità con ritardo a 4s.

5.2.2 Velocità (v_1) con parametri non default

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
10	3.0m	2.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.1s	0.1	0.2	0.7

Velocità t_1	Velocità t_2	Velocità t_3	Velocità t_4	Velocità t_5
25 m/s				

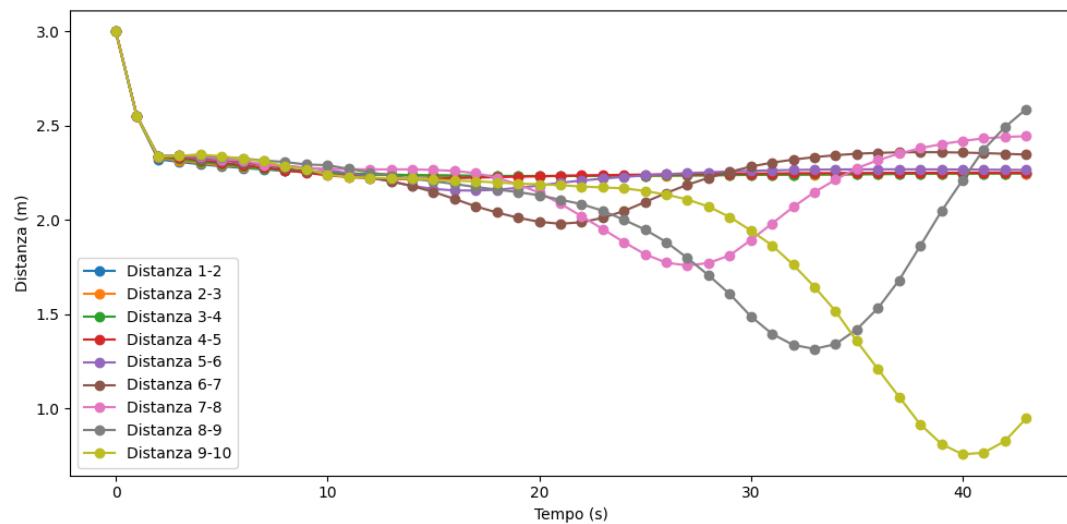


Figura 61: Distanze con velocità pilota costante a 25 m/s.

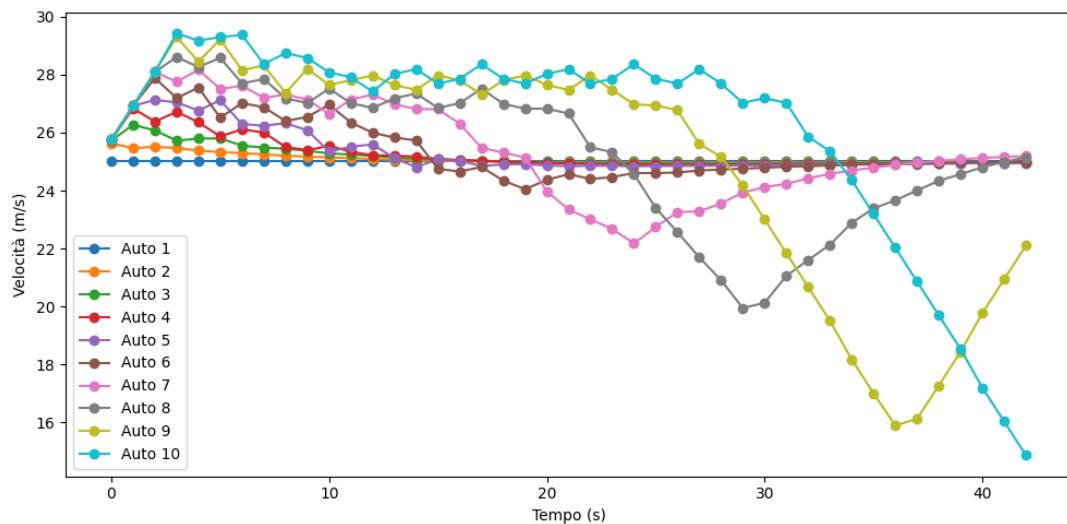


Figura 62: Velocità con velocità pilota costante a 25 m/s.

Esperimenti

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
10	3.0m	2.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.1s	0.1	0.2	0.7

Velocità t_1	Velocità t_2	Velocità t_3	Velocità t_4	Velocità t_5
25 m/s	25 m/s	15 m/s	25 m/s	25 m/s

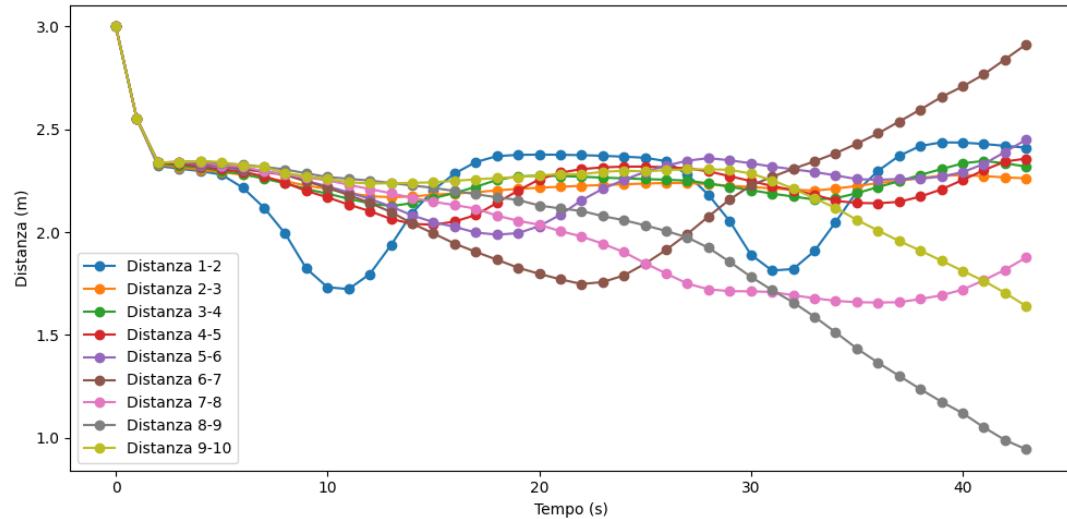


Figura 63: Distanze con velocità pilota variabile tra 15 m/s e 25 m/s.

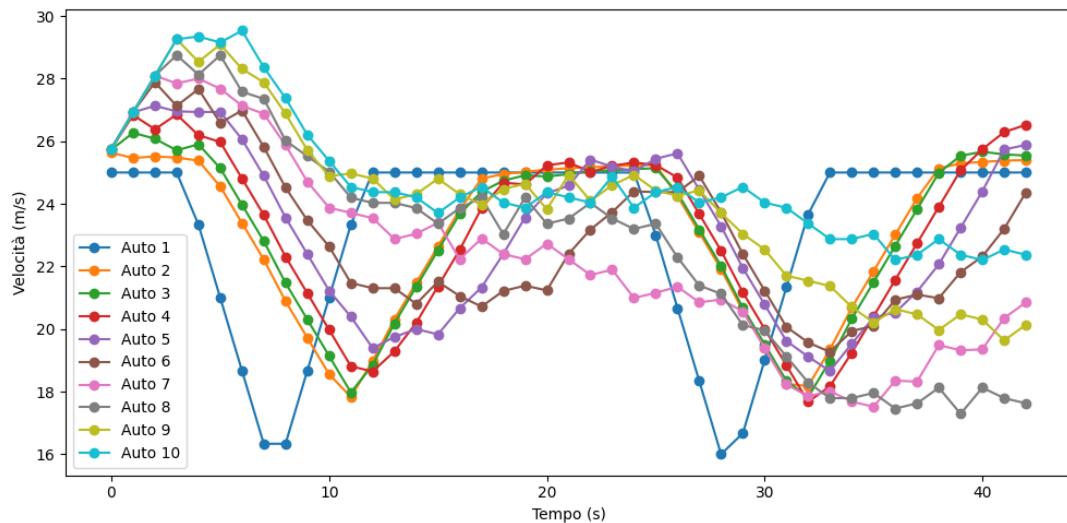


Figura 64: Velocità con velocità pilota variabile tra 15 m/s e 25 m/s.

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	3.0m	2.0m	0.2s
Time Headway (h)	Tau (τ)	k_p	k_d
0.05s	0.1	0.2	0.7

Velocità t_1	Velocità t_2	Velocità t_3	Velocità t_4	Velocità t_5
35 m/s	20 m/s	0 m/s	15 m/s	35 m/s

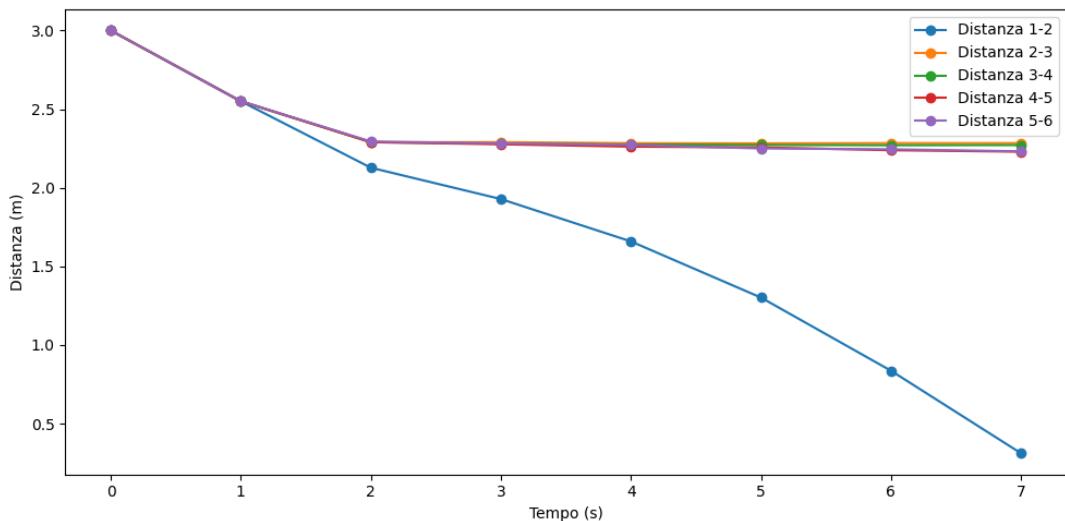


Figura 65: Distanze con velocità pilota variabile tra 0 m/s e 35 m/s.

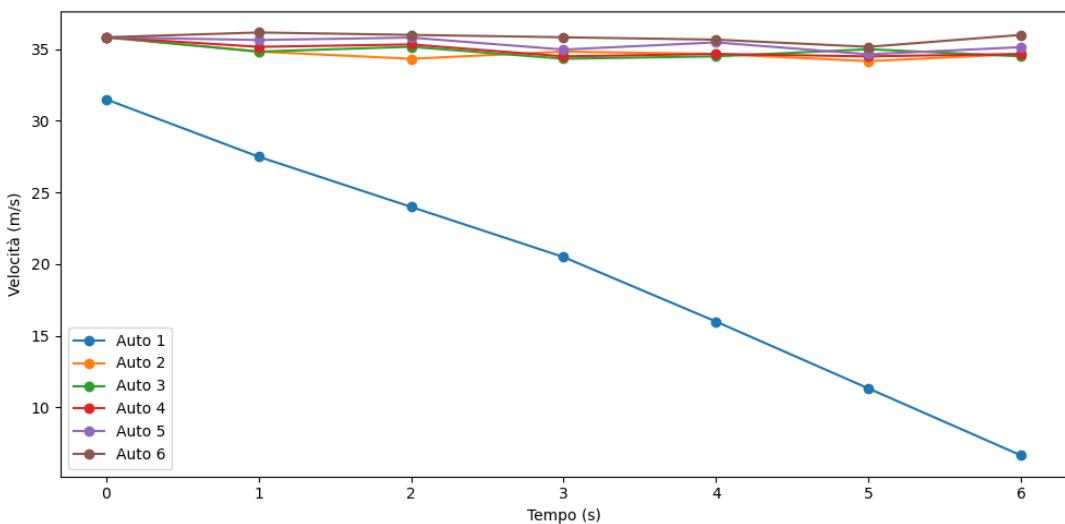


Figura 66: Velocità con velocità pilota variabile tra 0 m/s e 35 m/s.

Esperimenti

N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
7	4.5m	3.5m	1.0s
Time Headway (h)	Tau (τ)	k_p	k_d
0.3s	0.1	0.2	0.7

Velocità t_1	Velocità t_2	Velocità t_3	Velocità t_4	Velocità t_5
35 m/s	20 m/s	0 m/s	15 m/s	35 m/s

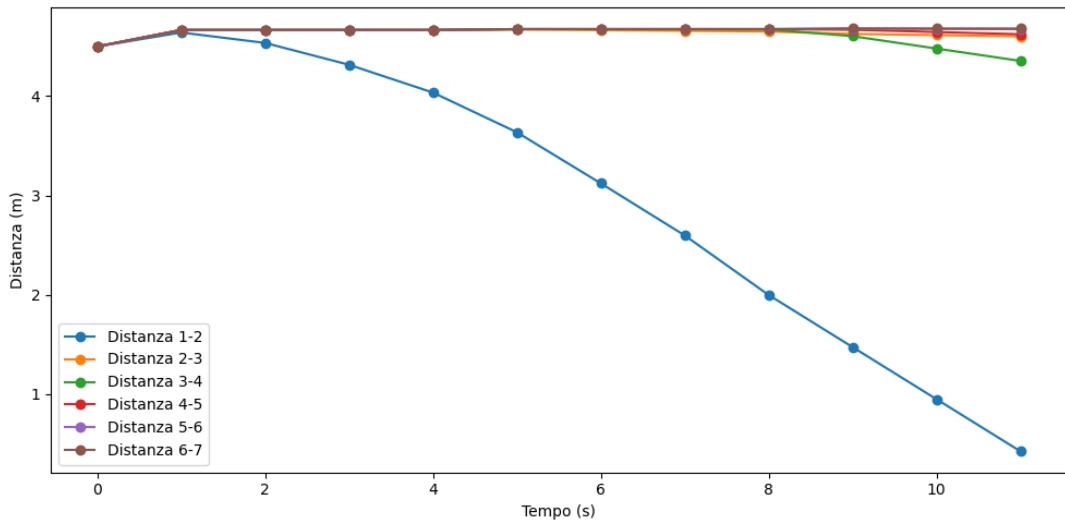


Figura 67: Distanze con velocità pilota variabile tra 0 m/s e 35 m/s.

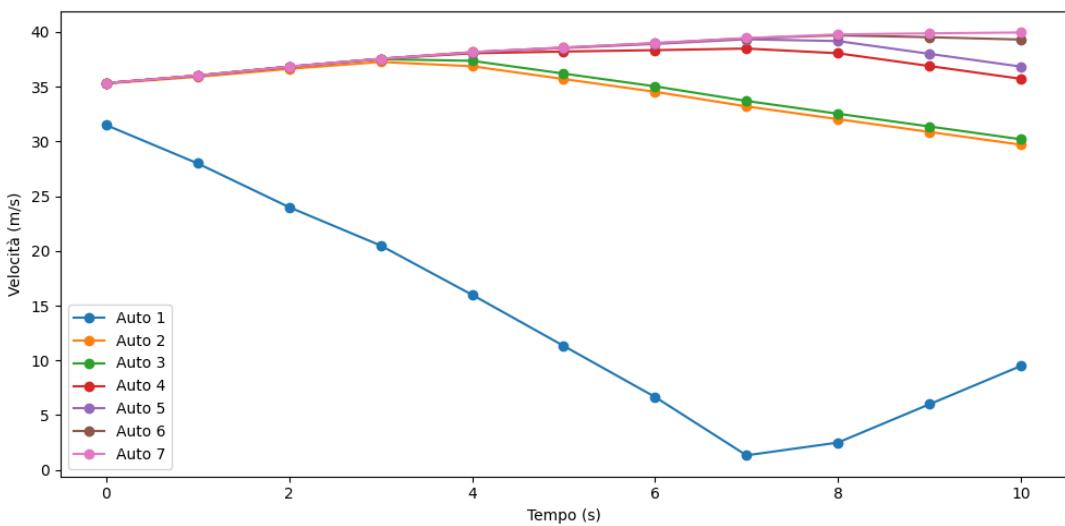


Figura 68: Velocità con velocità pilota variabile tra 0 m/s e 35 m/s.

6 Conclusioni

L'obiettivo di questo progetto è stato quello di creare un *frontend web* a scopo divulgativo capace di testare un modello di *platooning* CACC in maniera interattiva e dinamica. Il *focus* principale è stato quello di rendere l'interfaccia comprensibile a chiunque, così da poter agire come progetto di divulgazione ed informazione sulla materia del *platooning*; per questo l'applicazione è stata rilasciata su un dominio Vercel¹⁸ pubblico.

Per poter permettere una consultazione rapida del contenuto di questa relazione, lasciamo in allegato la presentazione [DM24] che riassume tutto il progetto in formato di diapositive. Oltre a questo, abbiamo anche reso disponibile pubblicamente tutto il codice usato nel progetto su una repository Github¹⁹; questa include tutti i csv e i grafici corrispondenti agli esperimenti riportati, il codice sorgente L^AT_EX di questa relazione e quello della presentazione allegata, entrambi compressi negli zip corrispondenti.

Nelle sezioni successive sono discusse le conclusioni relative agli esperimenti presenti in questa relazione e sono riportati esempi recenti di alcune applicazioni del *platooning* per comprenderne meglio i vantaggi e gli sviluppi pratici nel mondo reale.

6.1 Conclusioni degli esperimenti

Gli esperimenti indicano che i parametri della simulazione più significativi sono il ritardo di comunicazione, il *time headway* e i cambiamenti repentini di velocità da parte del veicolo pilota.

Nei modelli stabili si osserva che all'aumentare del ritardo di comunicazione tra i veicoli aumentano le oscillazioni delle velocità e delle distanze dovute a una risposta più lenta del sistema di controllo. Al contrario, un maggiore *time headway* riduce il rischio di collisioni, producendo curve più regolari nei grafici delle velocità e delle distanze. La variazione di τ intensifica o riduce la propagazione della velocità tra il primo e il secondo veicolo. Le variazioni di k_p influenzano le distanze e le velocità, mentre quelle di k_d hanno un impatto trascurabile sulle velocità misurate in sede di esperimento.

Inoltre, non c'è una propagazione degli errori dovuta al numero di veicoli, che mantiene la stabilità delle stringhe che abbiamo trattato nella sezione [2.3]. La distanza obiettivo, non fa osservare differenze significative nella di-

¹⁸ <https://platooning-simulation.vercel.app/>

¹⁹ <https://github.com/albbus-stack/platooning-simulation>

namica quando i parametri rimangono quelli di default; viene fatta eccezione nel caso in cui, con parametri non di default, sia impostata al minimo di $2m$; in questo caso può portare all'instabilità.

Infine, se variamo solo la velocità del veicolo pilota secondo una curva costante, il controllo si attenua e reagisce ai cambiamenti della vettura pilota nei veicoli successivi, come visibile nella sezione [5.1.9]. Tuttavia, se i cambiamenti nella velocità del veicolo pilota sono eccessivamente grandi e i parametri non sono quelli di default, questi possono rendere instabile il modello come mostrato nella sezione [5.2.2].

Riassumendo, il modello descritto non risente molto del cambiamento dei parametri di controllo k_p , k_d e τ . Sono dunque il *time headway* e il ritardo di comunicazione che hanno l'impatto maggiore sul sistema, vincolando il modello in modo più stringente nella risposta e nella propagazione degli errori nel plotone.

6.2 Applicazioni pratiche del platooning

Negli ultimi anni, soprattutto negli Stati Uniti, sono stati adottati e testati vari meccanismi di *platooning* per applicazioni sia industriali che civili. Un esempio recente è il rilascio in tre stati diversi di nuove flotte di veicoli automatizzati da parte dell'azienda *Kratos* (dettagliato nel comunicato stampa [KRT23]); in particolare il loro sistema è composto da un *leader-follower platoon*, cioè un plotone composto solo da due veicoli, il primo guidato da un operatore ed il secondo automatizzato. Questi veicoli usano il sistema ATMA (*Autonomous Truck Mounted Attenuator*) per proteggere da eventuali guidatori distratti gli operatori stradali che stanno compiendo manutenzioni a bassa velocità (asfaltatura o pittura della segnaletica orizzontale). Il funzionamento di questo meccanismo è osservabile nella dimostrazione video [KR19].

Un altro esempio, sempre applicato al campo civile, è quello di [FHW17], un progetto di ricerca esplorativa condotto dalla *Federal Highway Administration* che studia le applicazioni di sistemi di *platooning* per aumentare la capacità ed efficienza delle flotte di camion americani. In particolare sono stati condotti esperimenti proprio nel campo del trasporto pesante, come possiamo osservare dalla dimostrazione video [FHW18]. Oltre a questa applicazione, la FHWA ha anche testato dei sistemi CACC (*Cooperative Adaptive Cruise Control*), modelli analoghi a quello usato in questo progetto. Possiamo infatti vedere la dimostrazione video [FHW15] che mostra un sistema CACC, applicato ad automobili comuni, per gestire il traffico nelle strade a lunga percorrenza.

Bibliografia

- [PWN14] Jeroen Ploeg, Nathan van de Wouw e Henk Nijmeijer. «Lp String Stability of Cascaded Systems: Application to Vehicle Platooning». In: *IEEE Transactions on Control Systems Technology* 22 (mar. 2014), pp. 786–793. doi: 10.1109/tcst.2013.2258346.
- [FHW15] Federal Highway Administration FHWA. *Cooperative Adaptive Cruise Control (CACC)*. Video da YouTube²⁰. 2015.
- [AQ16] Autoscuola Quattroruote AQ. *Distanza di Sicurezza*. Immagine recuperata da autoscuola-quattroruote.com²¹. 2016.
- [FHW17] Federal Highway Administration FHWA. «Exploratory Advanced Research Program Expanding the Freight Capacity of America's Highways using Platooning and Connectivity to Increase Efficiency». In: FHWA-HRT-17-045 (lug. 2017). L'articolo è disponibile su fhwa.dot.gov²².
- [FHW18] Federal Highway Administration FHWA. *Partially Automated Truck Platooning Demonstration*. Video da YouTube²³. 2018.
- [KR19] Murad Al Qurishee KR. *Autonomous Truck Mounted Attenuator (ATMA) test*. Video da YouTube²⁴. 2019.
- [PO21] Patentino Online PO. *Distanza di Sicurezza*. Immagini recuperate da patentinoonline.it²⁵. 2021.
- [KRT23] Kratos *leader-follower platoon (ATMA)* KRT. *Kratos Defense Self-Driving Trucks are on the Road Across the United States, Increasing Worker Safety and Addressing Workforce Shortfalls*. Il comunicato stampa è disponibile su ir.kratosdefense.com²⁶. 2023.
- [DM24] Alberto Del Buono Paolini e Federico Marra. «Frontend per la simulazione di un problema di platooning». In: Presentazione allegata a questo documento di tesi, reperibile digitalmente qui²⁷. Firenze, Italia, apr. 2024.

²⁰ https://www.youtube.com/watch?v=D_2DPm9v-Lw

²¹ <https://www.autoscuola-quattroruote.com/portfolio/distanza-di-sicurezza/>

²² <https://www.fhwa.dot.gov/publications/research/ear/17045/17045.pdf>

²³ <https://www.youtube.com/watch?v=iNTKqh7i5jQ>

²⁴ <https://www.youtube.com/watch?v=CWypqlNewMU>

²⁵ <https://www.patentinoonline.it/distanza-di-sicurezza>

²⁶ <https://ir.kratosdefense.com/news-releases/news-release-details/kratos-defense-self-driving-trucks-are-road-across-united-states>

²⁷ <http://platooning-simulation.vercel.app/pdf/presentation.pdf>

Elenco delle figure

1	Distanza di sicurezza [PO21]	1
2	Componenti distanza di sicurezza [AQ16]	2
3	Spazio di reazione [PO21]	3
4	<i>Platoon</i> di veicoli equipaggiati con CACC	4
5	Visualizzazione grafica della simulazione.	10
6	Varie impostazioni numeriche per la simulazione.	17
7	Grafico per impostare l'andamento del primo veicolo del convoglio nel tempo.	18
8	Interfaccia utente per l'apertura dei pannelli (<i>Sliver</i>), con sopra annotate le scorciatoie da tastiera.	18
9	Grafico della distanza tra due veicoli del convoglio.	20
10	Grafico della velocità di un veicolo del convoglio.	21
11	Bottone di esportazione dei dati della simulazione in csv.	21
12	Percorsi dinamici corrispondenti ad inglese (/), italiano (/it) e francese (/fr).	24
13	Distanze con parametri di default.	30
14	Velocità con parametri di default.	30
15	Distanze con 8 auto.	31
16	Velocità con 8 auto.	31
17	Distanze con 10 auto.	32
18	Velocità con 10 auto.	32
19	Distanze con 2.0m come distanza obiettivo.	33
20	Velocità con 2.0m come distanza obiettivo.	33
21	Distanze con 20.0m come distanza obiettivo.	34
22	Velocità con 20.0m come distanza obiettivo.	34
23	Distanze con ritardo a 0.1s.	35
24	Velocità con ritardo a 0.1s.	35
25	Distanze con ritardo a 0.5s.	36
26	Velocità con ritardo a 0.5s.	36
27	Distanze con ritardo a 0.7s.	37
28	Velocità con ritardo a 0.7s.	37
29	Distanze con ritardo a 1s.	38
30	Velocità con ritardo a 1s.	38
31	Distanze con <i>time headway</i> a 0.1s.	39
32	Velocità con <i>time headway</i> a 0.1s.	39

33	Distanze con <i>time headway</i> a 1s.	40
34	Velocità con <i>time headway</i> a 1s.	40
35	Distanze con <i>time headway</i> a 2s.	41
36	Velocità con <i>time headway</i> a 2s.	41
37	Distanze con τ a 0.01.	42
38	Velocità con τ a 0.01.	42
39	Distanze con τ a 0.7.	43
40	Velocità con τ a 0.7.	43
41	Distanze con k_p a 0.01.	44
42	Velocità con k_p a 0.01.	44
43	Distanze con k_p a 1.5.	45
44	Velocità con k_p a 1.5.	45
45	Distanze con k_d a 0.1.	46
46	Velocità con k_d a 0.1.	46
47	Distanze con k_d a 1.5.	47
48	Velocità con k_d a 1.5.	47
49	Distanze con velocità pilota costante a 10 m/s.	48
50	Velocità con velocità pilota costante a 10 m/s.	48
51	Distanze con velocità pilota costante a 35 m/s.	49
52	Velocità con velocità pilota costante a 35 m/s.	49
53	Distanze con velocità pilota variabile tra 20 m/s e 0 m/s.	50
54	Velocità con velocità pilota variabile tra 20 m/s e 0 m/s.	50
55	Distanze con velocità pilota variabile tra 35 m/s e 0 m/s.	51
56	Velocità con velocità pilota variabile tra 35 m/s e 0 m/s.	51
57	Distanze con ritardo a 3s.	53
58	Velocità con ritardo a 3s.	53
59	Distanze con ritardo a 4s.	54
60	Velocità con ritardo a 4s.	54
61	Distanze con velocità pilota costante a 25 m/s.	55
62	Velocità con velocità pilota costante a 25 m/s.	55
63	Distanze con velocità pilota variabile tra 15 m/s e 25 m/s.	56
64	Velocità con velocità pilota variabile tra 15 m/s e 25 m/s.	56
65	Distanze con velocità pilota variabile tra 0 m/s e 35 m/s.	57
66	Velocità con velocità pilota variabile tra 0 m/s e 35 m/s.	57
67	Distanze con velocità pilota variabile tra 0 m/s e 35 m/s.	58
68	Velocità con velocità pilota variabile tra 0 m/s e 35 m/s.	58