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

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, ma anche a un aumento della durata degli spostamenti, con conseguenti ritardi, stress per gli automobilisti e impatti negativi sull'ambiente 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 convogliamento, consiste nell'allineamento e nella gestione coordinata di più veicoli in movimento lungo una strada. Questa formazione di veicoli segue un veicolo guida, cercando di mantenere una distanza costante prestabilita e ridotta tra di loro e minore in confronto alla distanza di si, e risponde 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.
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 convoglio, garantendo contemporaneamente il rispetto delle normative stradali e la massima sicurezza per i conducenti e gli altri utenti della strada.
A questo scopo, il controllo di crociera adattivo e cooperativo (CACC) si presenta come una soluzione 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 wireless tra veicoli, il CACC consente di mantenere intervalli di tempo significativamente inferiori rispetto ai sistemi di controllo di crociera adattivi tradizionali, contribuendo così a aumentare la capacità delle strade e a ridurre il consumo di carburante.
Da annoverare fra i vantaggi 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:

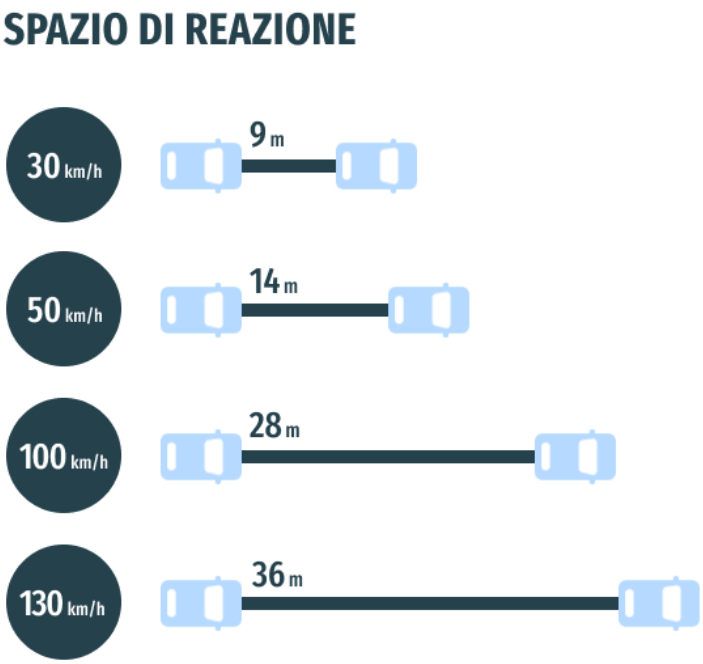

Figura 1: Spazio di reazione di un veicolo con guidatore umano con tempo di reazione $1s$ [Qua16]

quindi considerando anche lo spazio di frenatura come mostrato in figura

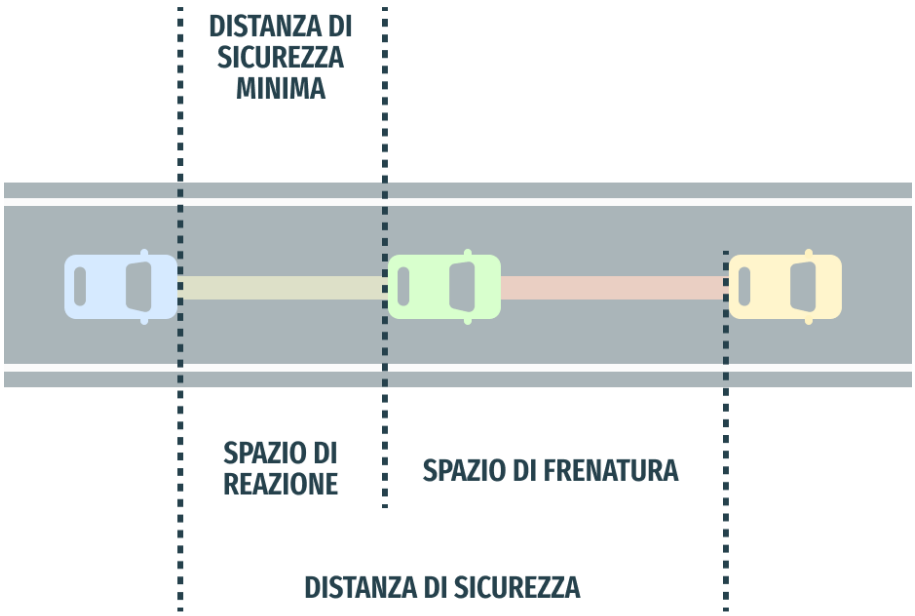

Figura 2: Componenti distanza di sicurezza [Onl21]

Tuttavia, nonostante i suoi evidenti vantaggi, l'implementazione efficace del CACC e dei sistemi di platooning richiede la progettazione e lo sviluppo di un frontend sofisticato e efficiente per la simulazione e la valutazione delle prestazioni di tali sistemi. È qui che il nostro progetto entra in gioco. Ci proponiamo di sviluppare un frontend innovativo per la simulazione del platooning di veicoli, concentrandoci sull'ottimizzazione del modello fisico e del sistema di controllo, al fine di massimizzare l'efficienza del traffico stradale, garantendo al contempo la massima sicurezza per tutti gli utenti della strada.

Figura 3: Distanza di sicurezza [Qua16]

2 Platooning di veicoli

Utilizziamo il metodo descritto nell'articolo [PWN14].

2.1 Modello matematico del platooning

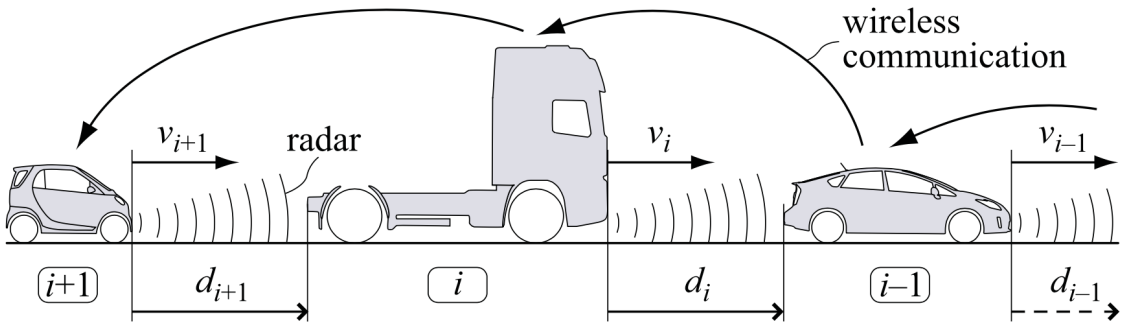

Figura 4: Platoon di veicoli equipaggiati con CACC

2.2 Equazioni in forma matriciale

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

Riassunta: $\dot{x}_i = A_0 x_i + A_1 x_{i-1}$ con $x_i = (e_i, v_i, a_i, u_i)^t$ che rappresenta il vettore allo stato i.
Svolgiamo il prodotto fra matrici trascurando il fattore k_{dd} che poniamo a 0 e otteniamo:

L'errore: $\dot{e}_i = -v_i - ha_i + v_{i-1}$

La velocità: $\dot{v}_i = a_i$

L'accelerazione: $\dot{a}_i = -\frac{a_i}{\tau} + \frac{u_i}{\tau} = \frac{-a_i+u_i}{\tau}$

Il controllo: $\dot{u}_i = \frac{k_p}{h}e_i - \frac{k_d}{h}v_i - k_d a_i - \frac{u_i}{h} + \frac{k_d}{h}v_{i-1} + \frac{u_{i-1}}{h} = \frac{k_p e_i - k_d v_i - u_i + k_d v_{i-1} + u_{i-1}}{h} - k_d a_i$

2.3 Campionamento e discretizzazione delle equazioni

3 Simulazione del platooning

3.1 Implementazione della simulazione

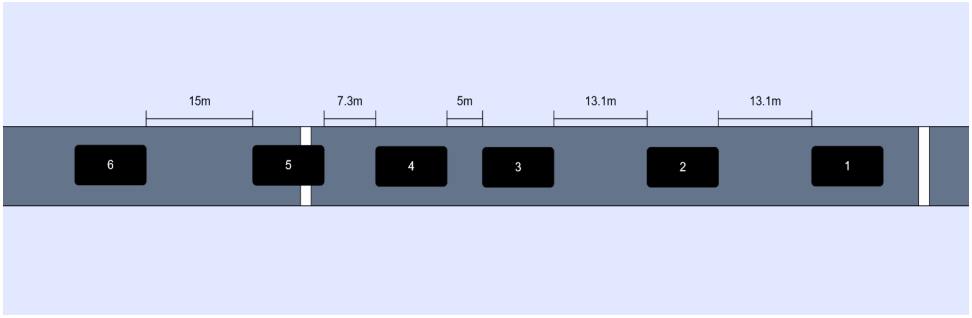

Figura 5: Visualizzazione grafica della simulazione.

3.2 Rappresentazione software delle equazioni del platooning

Funzione draw() in P5Canvas.tsx

```
// Setup for the first car
prevV[0] = velocity[0];
prevU[0] = controlU[0];
acceleration[0] = leadingCarChart[(leadingCarChartIndex + 1) %
    leadingCarChart.length].velocity - leadingCarChart[leadingCarChartIndex].velocity;
velocity[0] += acceleration[0] / FS;
controlU[0] = acceleration[0];
error[0] = 0;

const standstillDistance = carSpacing * 10 + CAR_WIDTH;

// Update other cars values
for (let i = 1; i < carNumber; i++) {
  // Time i-1 (previous time)
  const prevE = error[i];
  const prevA = acceleration[i];

  // Time i (actual time). Divide Δei, Δvi, Δai, Δui by the simulation frequency
  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;

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

  // Limit the distance step & update car positions
  const maxStep = 1;
  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;
    }
  }
  carPoints[i] = carPoints[i - 1] - d;

  // Update previous velocity & control values
  prevV[i] = velocity[i];
  prevU[i] = controlU[i];
}
```

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 la frontend è accessibile pubblicamente su platooning-visualization.vercel.app[3]. Abbiamo optato per ospitare l'app su Vercel[4] 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. Nelle sezioni successive ne saranno definite altre in corrispondenza col loro utilizzo.

[1] https://nextjs.org
[2] https://github.com/albbus-stack/platooning-visualization
[3] https://platooning-visualization.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 obbiettivo tra di loro*, *Distanza temporale tra di loro*, *Delay di comunicazione* ed alcuni parametri del modello di platooning precedentemente discusso come *Tau*, *Kp* e *Kd*.

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

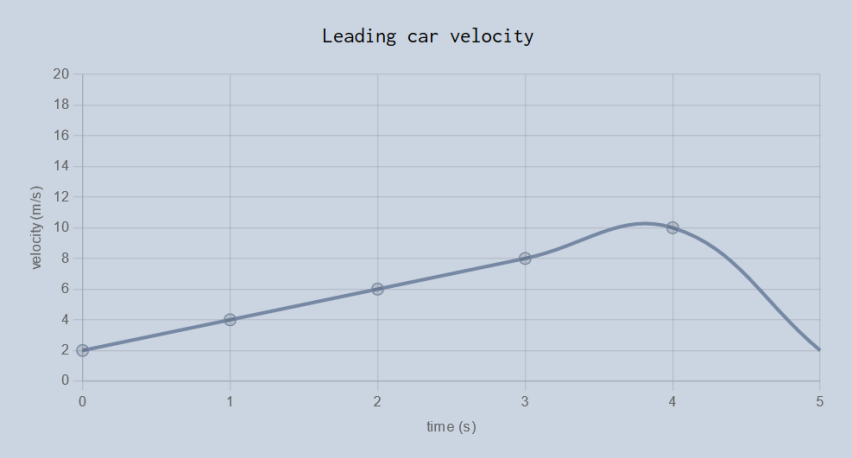

Figura 7: Grafico per impostare l'andamento del primo veicolo del convoglio nel tempo.

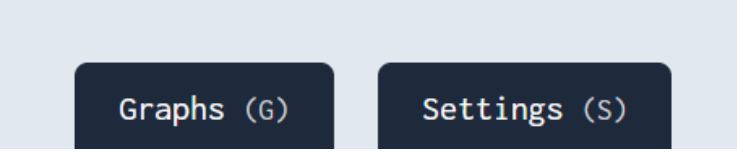

Figura 8: Interfaccia utente per l'apertura dei pannelli (*Sliver*), con sopra annotate le scorciatoie da tastiera.

La velocità del primo veicolo del convoglio è controllabile tramite un grafico interattivo, in figura 7, in cui sono modificabili i punti da $t = 0$ a $t = 4$. Il primo veicolo 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*).

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 slider.

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 *contesto 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 *60 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 di distanza o velocità specifico per un veicolo; questi sono selezionabili da due dropdown presenti sopra ai grafici corrispondenti.

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

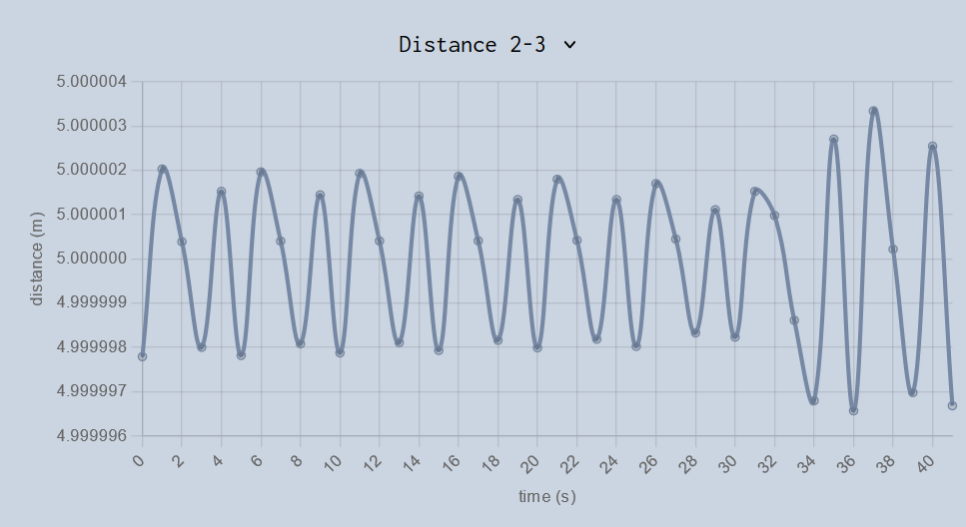

Figura 9: Grafico della distanza tra due veicoli del convoglio.

[9] https://www.chartjs.org

[10] https://react-chartjs-2.js.org

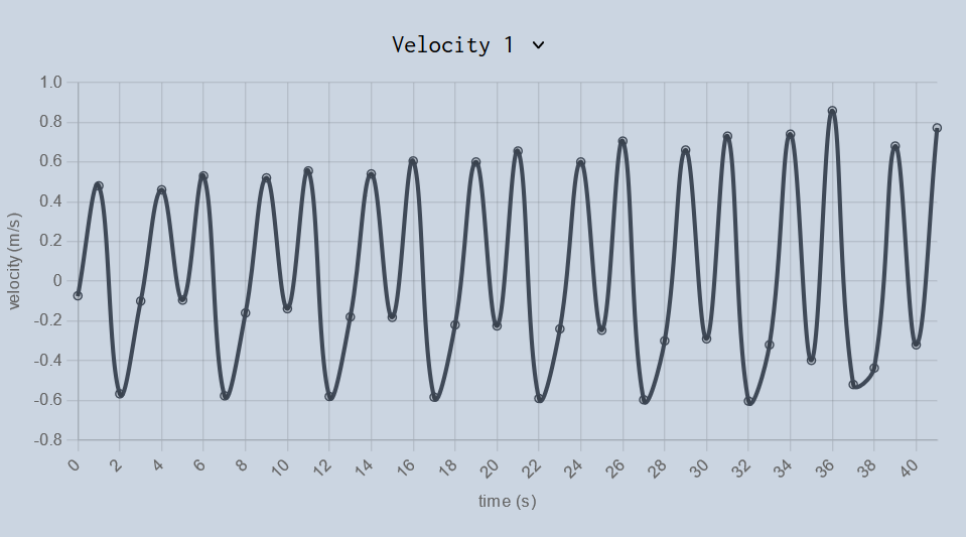

Figura 10: Grafico della velocità di un veicolo del convoglio.

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 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 alcuni di questi *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 localizzazioni: *en*, *it*, *fr*, *es*, *de*, *dk*, *nl*, *pt*, *ru*, *sa*, *in*, *cn*, *jp*, *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].

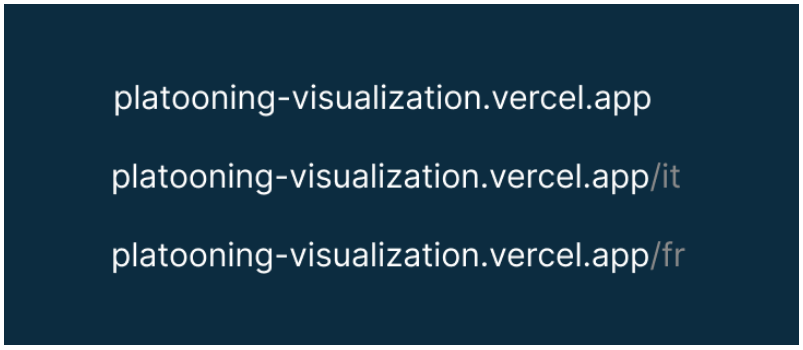

Figura 12: Percorsi dinamici corrispondenti ad inglese (`/`), italiano (`/it`) e francese (`/fr`).

[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-visualization",
  "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 loro 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-visualization",
  "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.

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)
  plt.close()
```

[16] https://github.com/albbus-stack/platooning-visualization/tree/main/experiments

5.1 Modello stabile

5.1.1 Parametri di default

Questi parametri sono presi dall'articolo [PWN14].

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

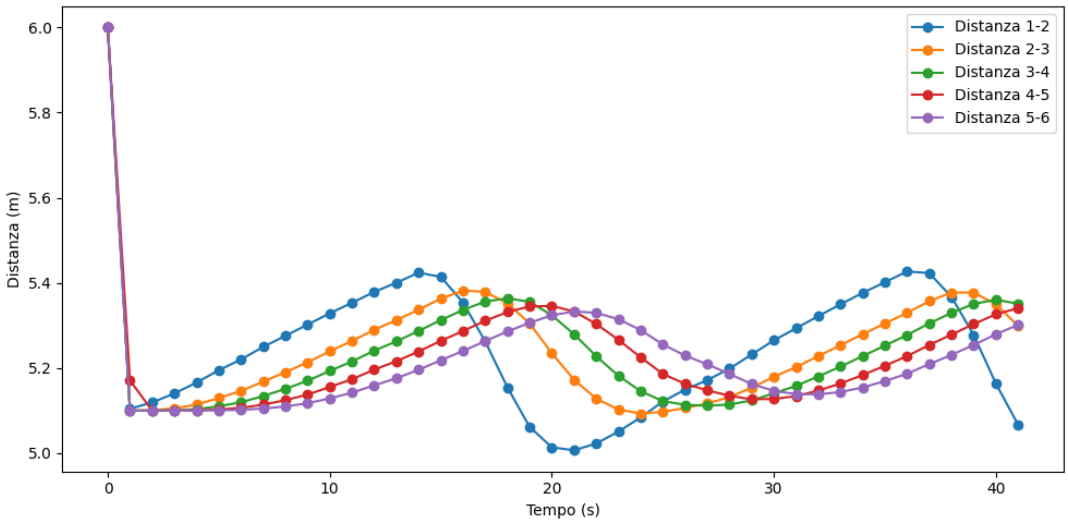

Figura 13: Distanze con parametri di default.

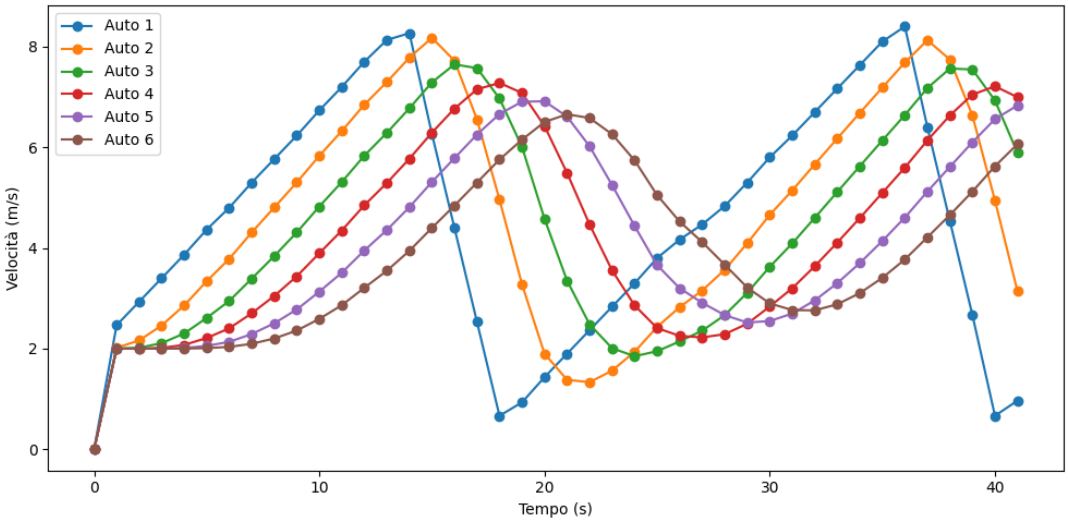

Figura 14: Velocità con parametri di default.

5.1.2 Numero di auto

N° auto	Distanza Iniziale	Distanza Target	Ritardo
3	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

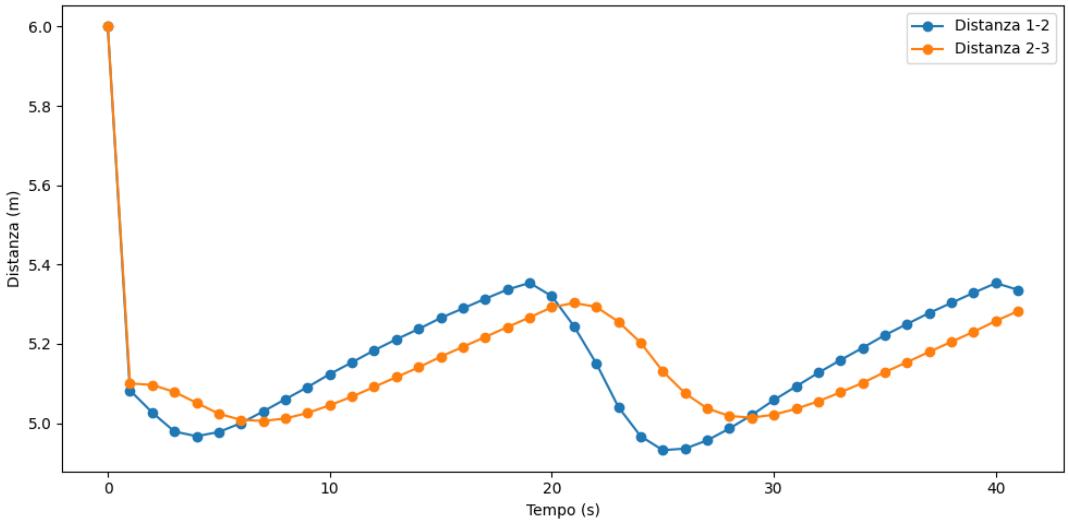

Figura 15: Distanze con 3 auto.

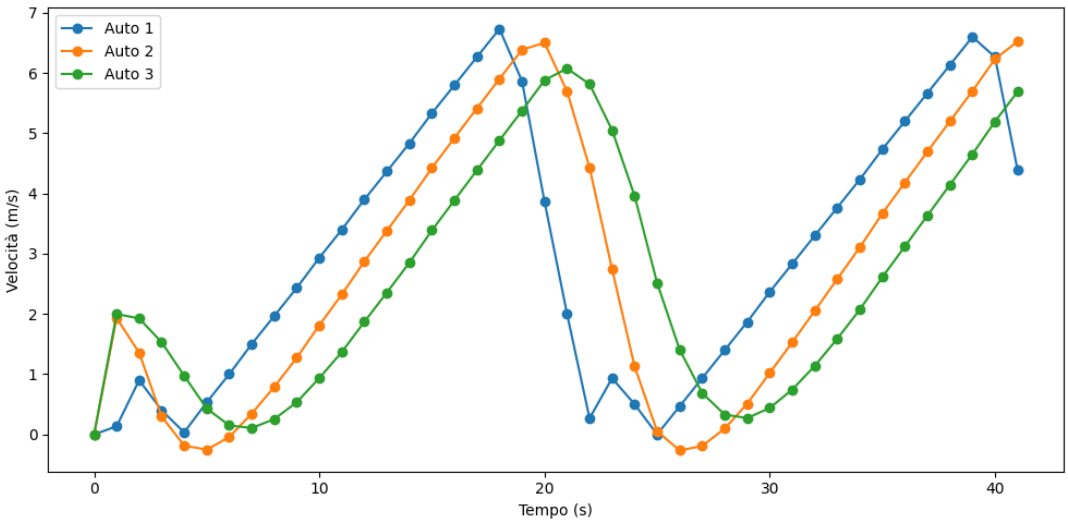

Figura 16: Velocità con 3 auto.

N° auto	Distanza Iniziale	Distanza Target	Ritardo
8	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

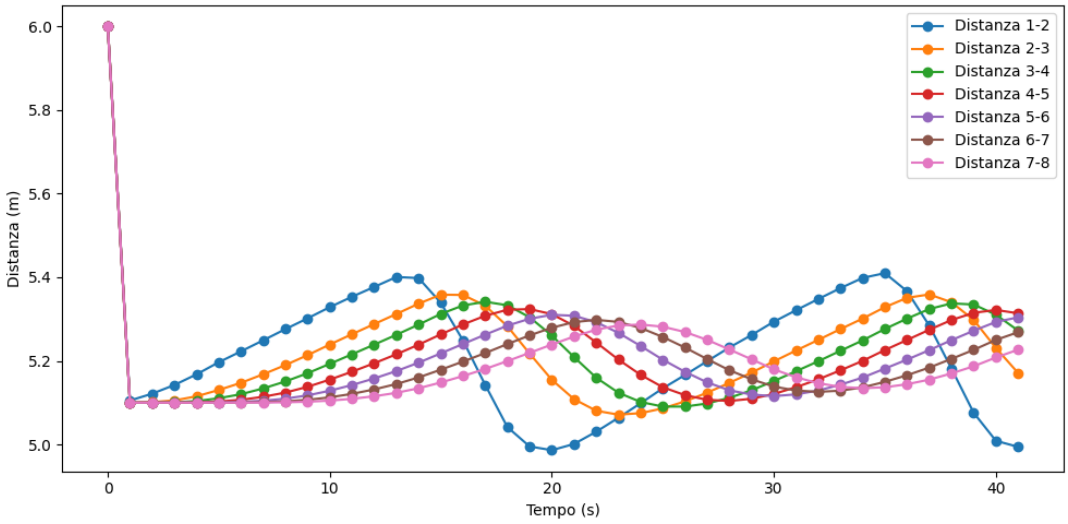

Figura 17: Distanze con 8 auto.

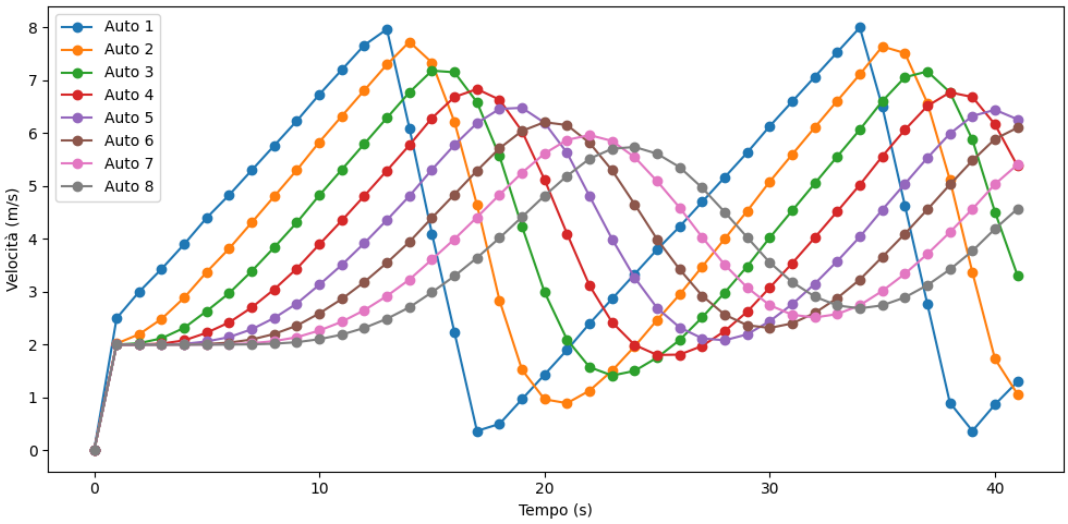

Figura 18: Velocità con 8 auto.

N° auto	Distanza Iniziale	Distanza Target	Ritardo
10	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

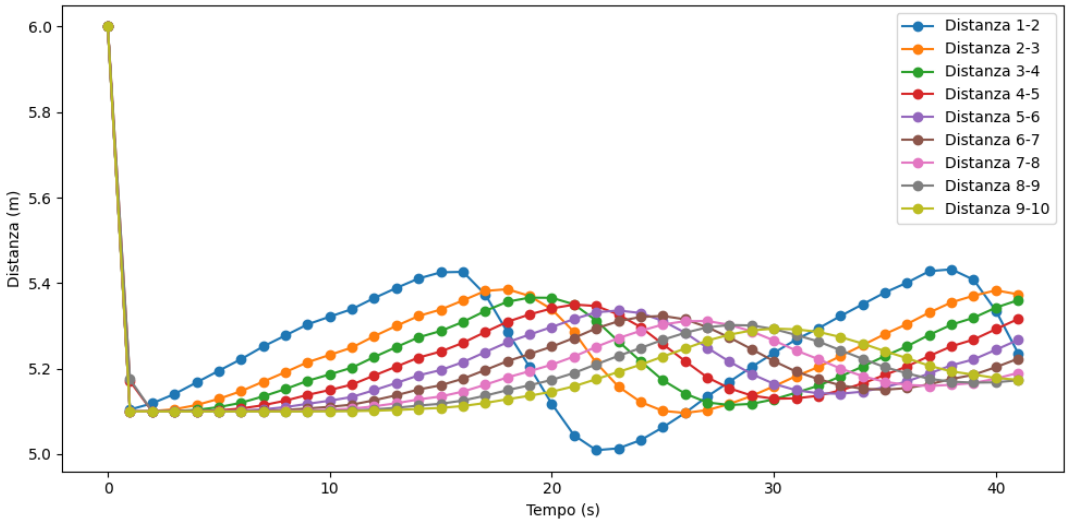

Figura 19: Distanze con 10 auto.

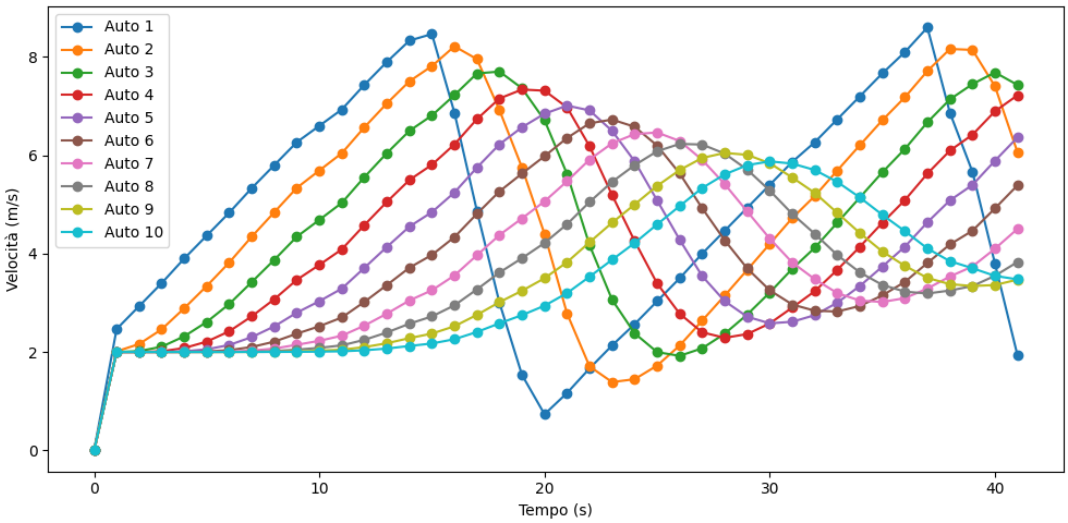

Figura 20: Velocità con 10 auto.

5.1.3 Spazio tra le auto

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$2.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

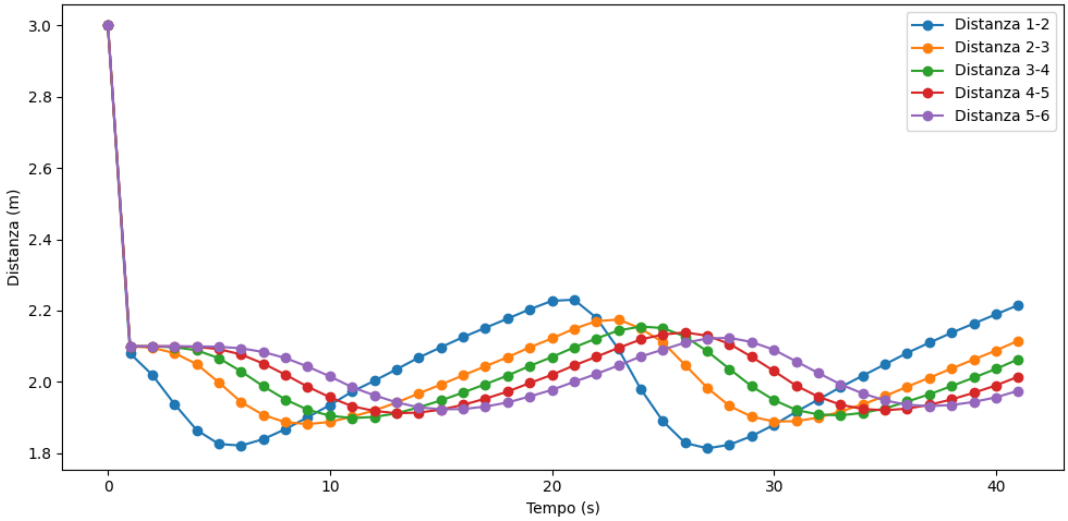

Figura 21: Distanze con $2.0m$ come distanza iniziale.

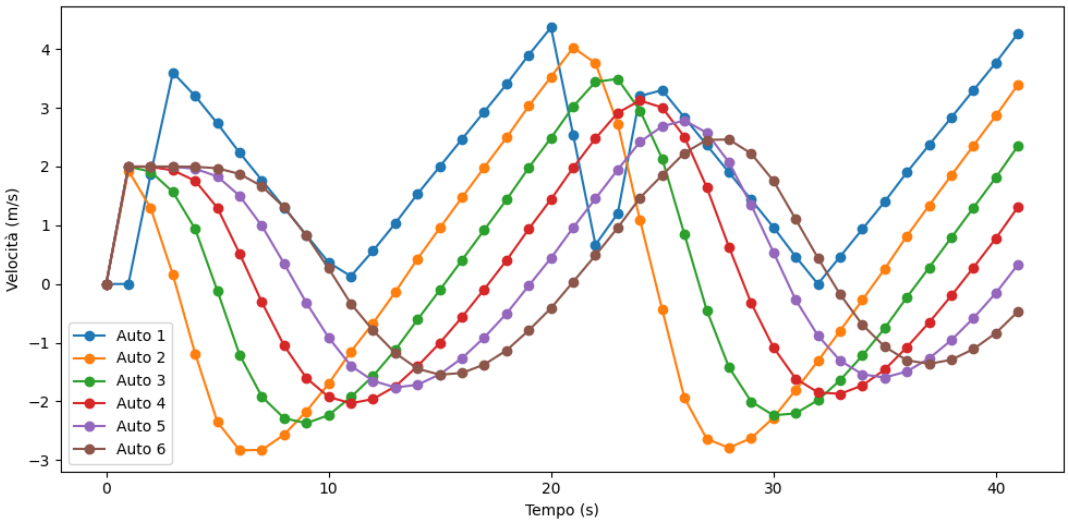

Figura 22: Velocità con distanza iniziale $2.0m$.

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$10.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

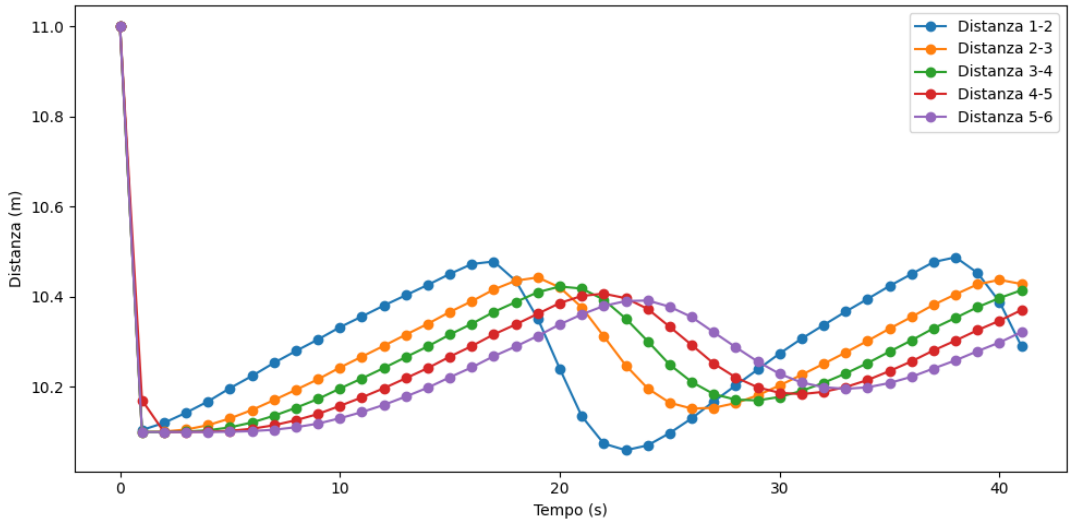

Figura 23: Distanze con $10.0m$ come distanza iniziale.

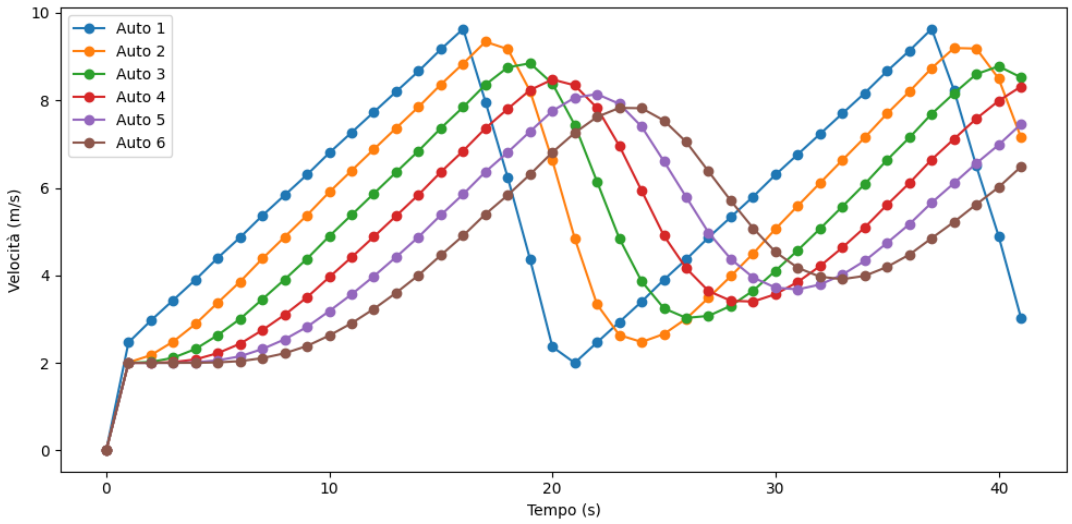

Figura 24: Velocità con distanza iniziale $10.0m$.

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$20.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

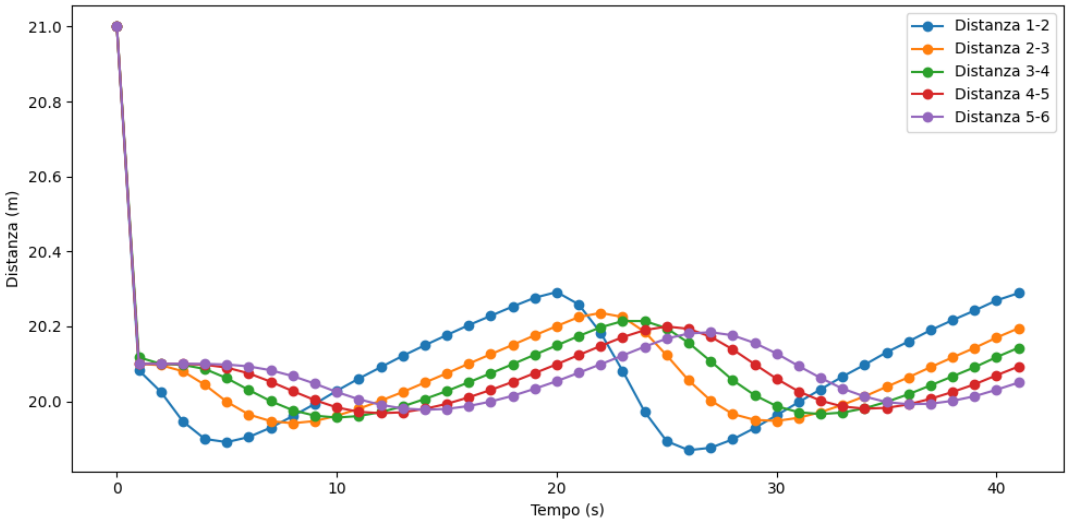

Figura 25: Distanze con $20.0m$ come distanza iniziale.

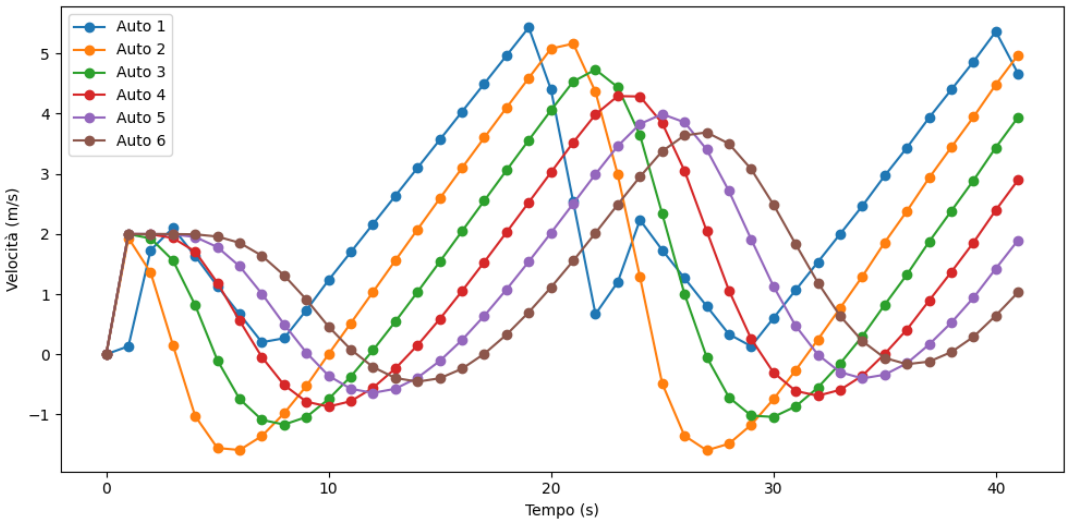

Figura 26: Velocità con distanza iniziale $20.0m$.

5.1.4 Ritardo di comunicazione

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$0.5s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

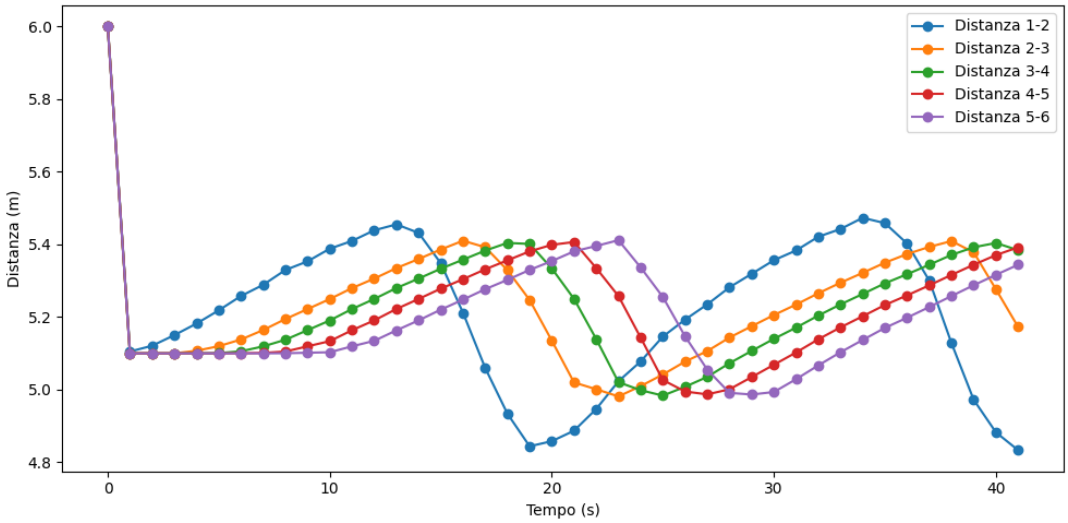

Figura 27: Distanze con ritardo a $0.5s$.

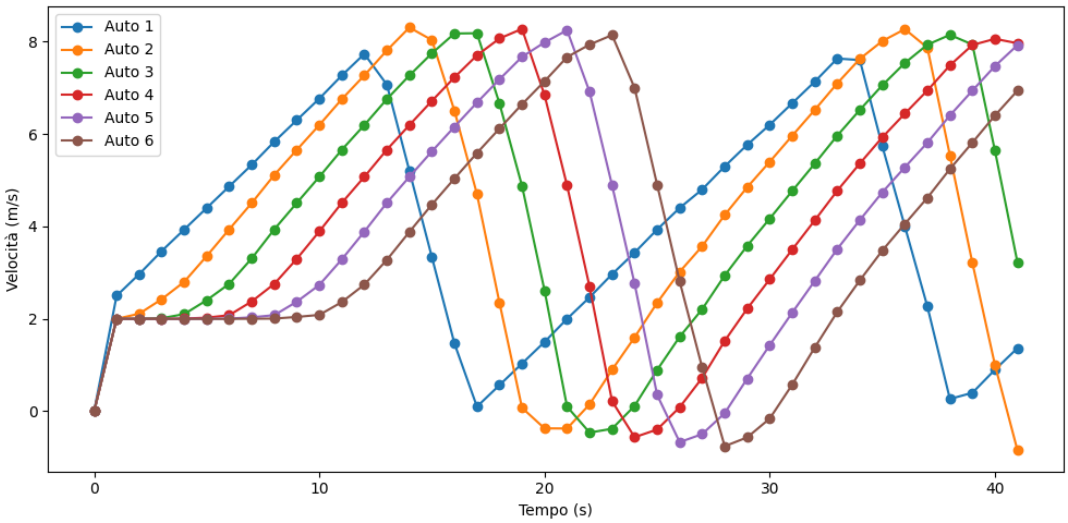

Figura 28: Velocità con ritardo a $0.5s$.

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$1s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

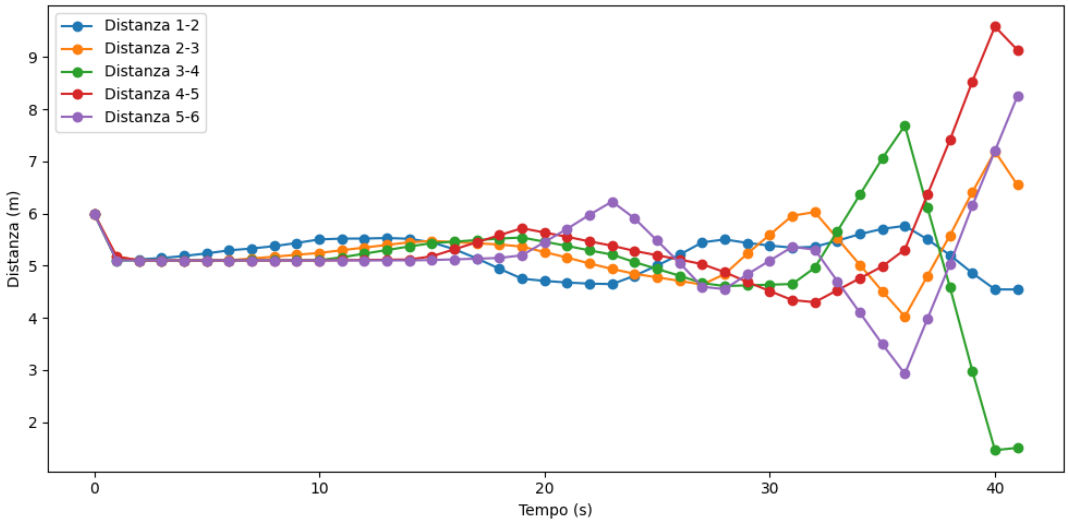

Figura 29: Distanze con ritardo a $1s$.

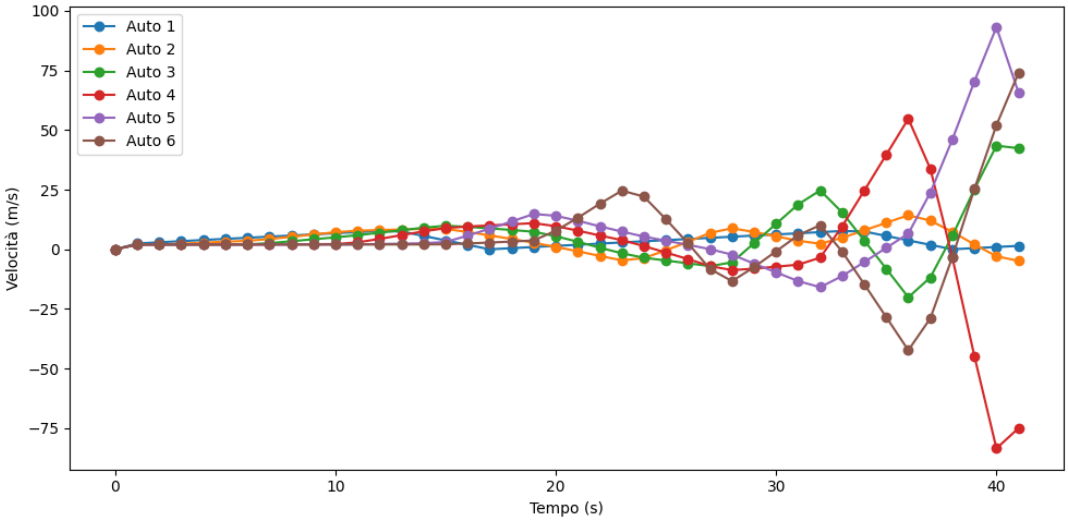

Figura 30: Velocità con ritardo a $1s$.

5.1.5 Time Headway

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.05	0.1	0.2	0.7

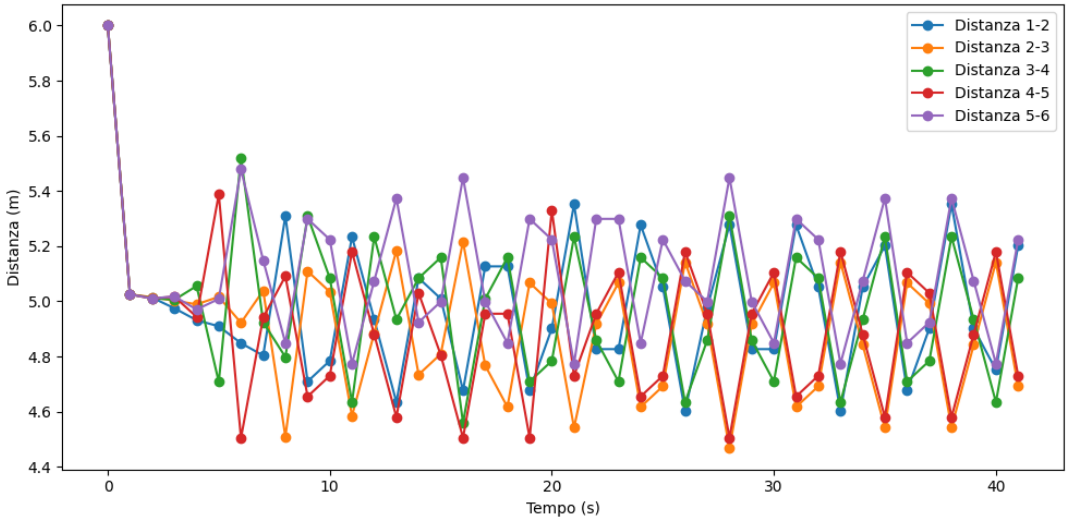

Figura 31: Distanze con $Time\ Headway$ a 0.05.

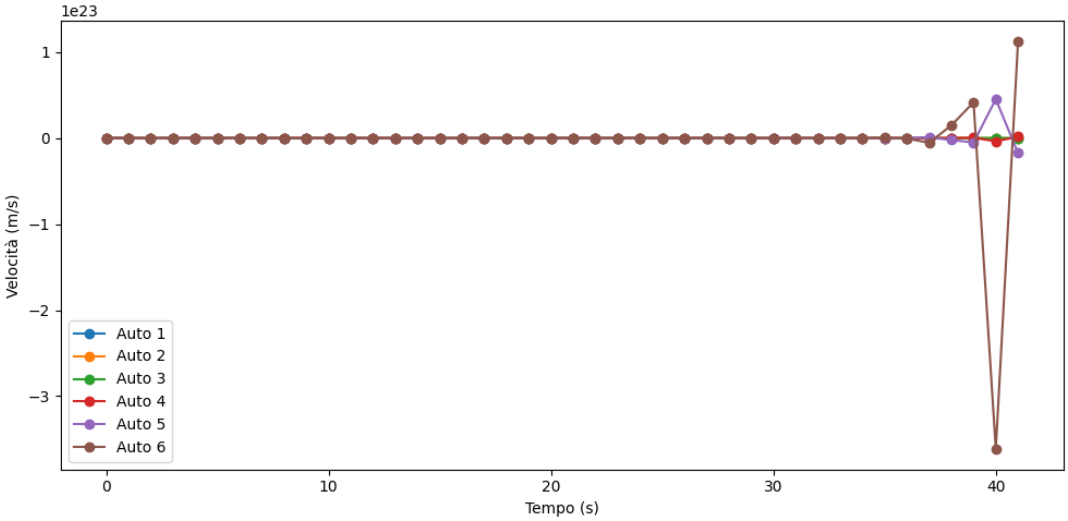

Figura 32: Velocità con $Time\ Headway$ a 0.05.

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.1	0.1	0.2	0.7

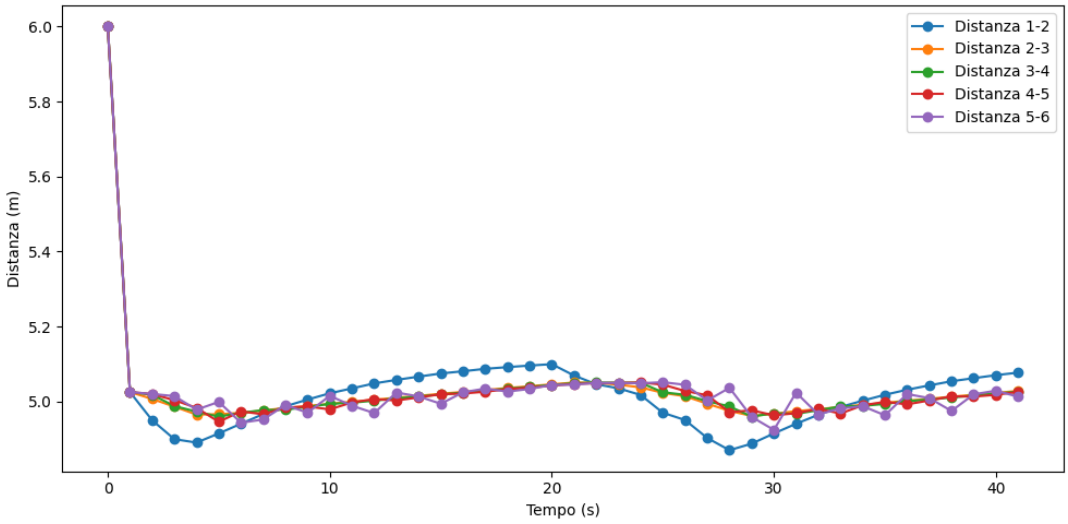

Figura 33: Distanze con $Time\ Headway$ a 0.1.

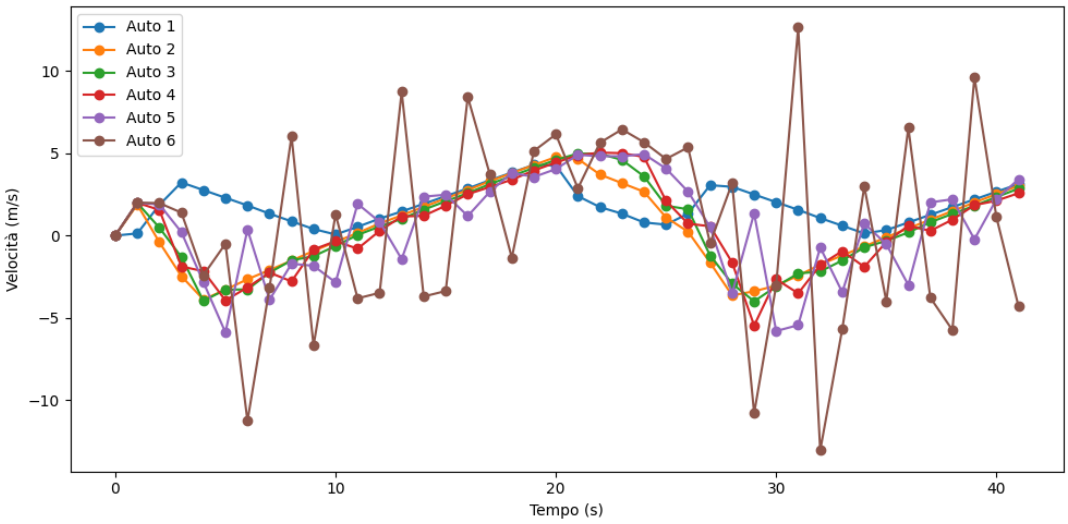

Figura 34: Velocità con $Time\ Headway$ a 0.1.

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
1	0.1	0.2	0.7

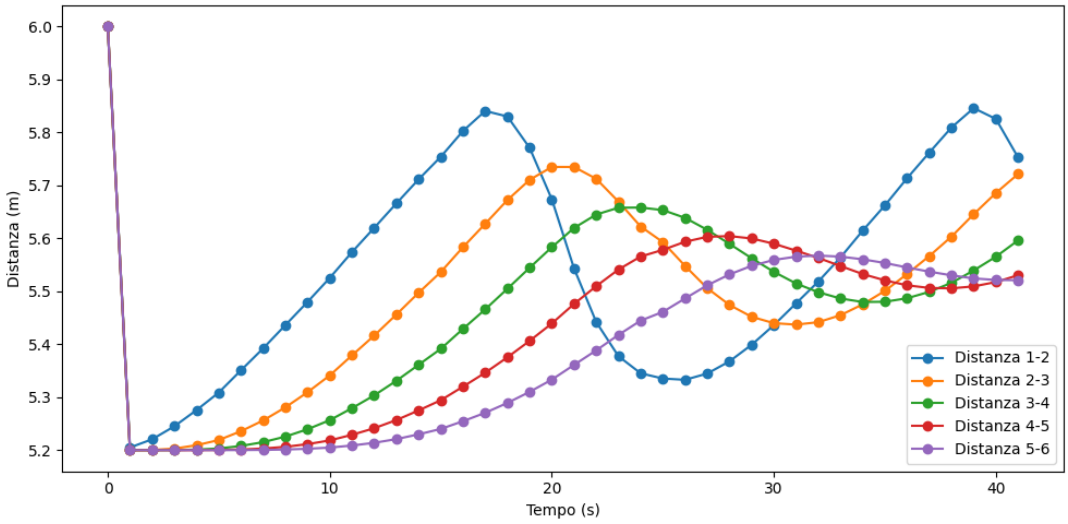

Figura 35: Distanze con $Time\ Headway$ a 1.

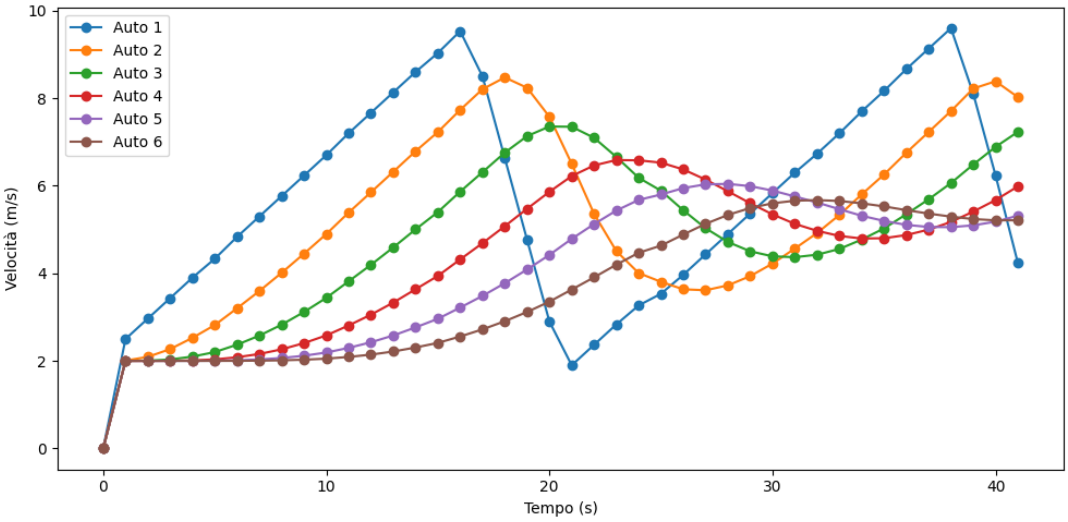

Figura 36: Velocità con $Time\ Headway$ a 1.

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
2	0.1	0.2	0.7

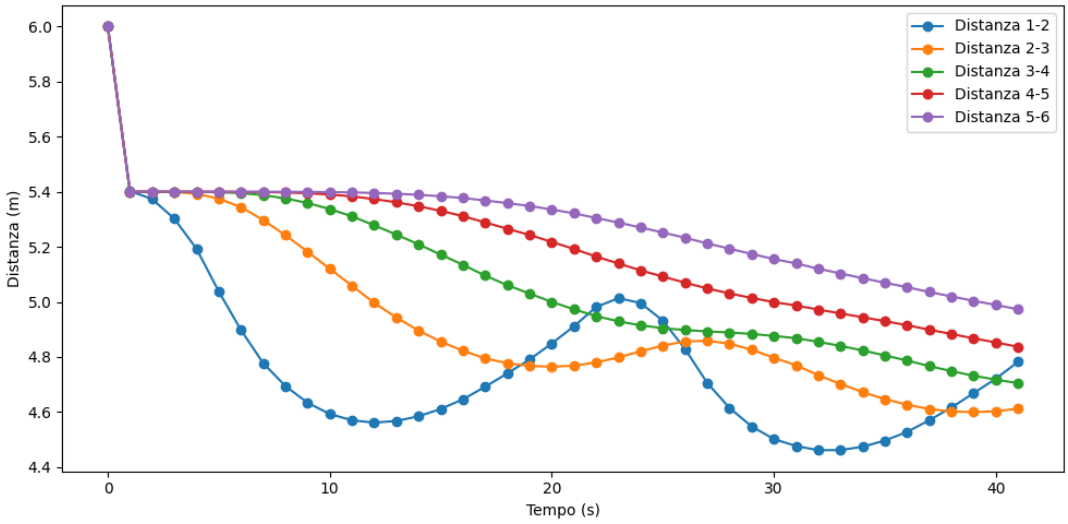

Figura 37: Distanze con $Time\ Headway$ a 2.

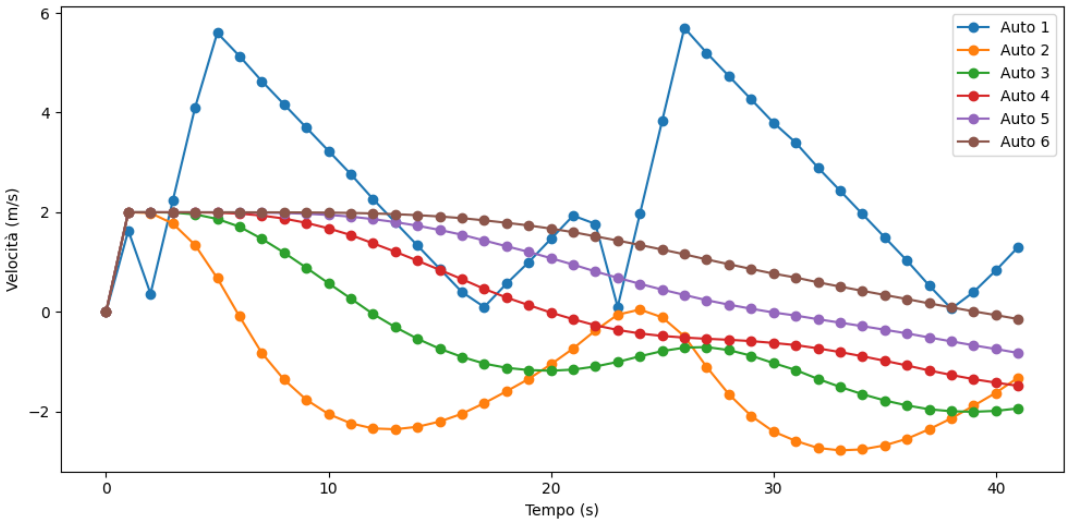

Figura 38: Velocità con $Time\ Headway$ a 2.

5.1.6 Tau (τ)

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.01	0.2	0.7

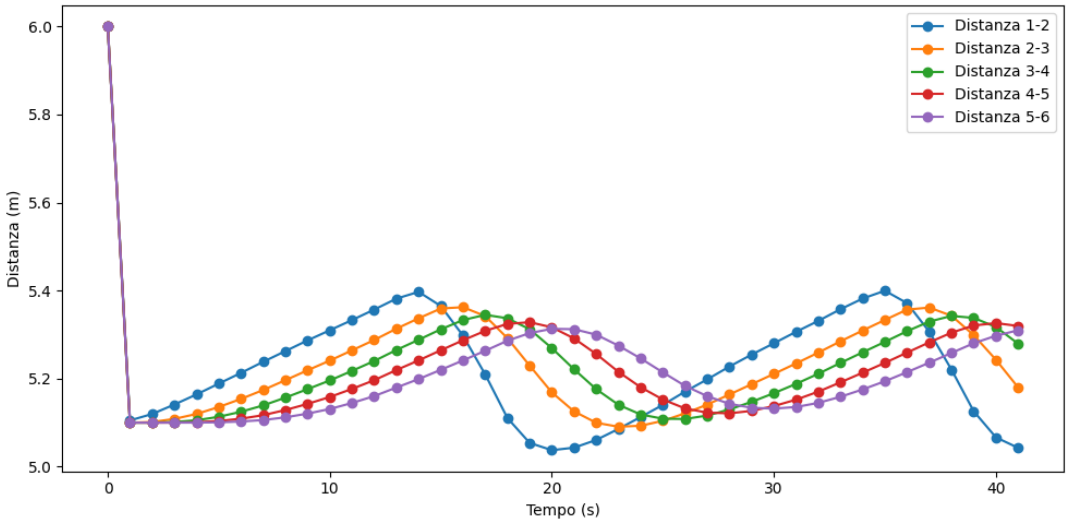

Figura 39: Distanze con τ a 0.01.

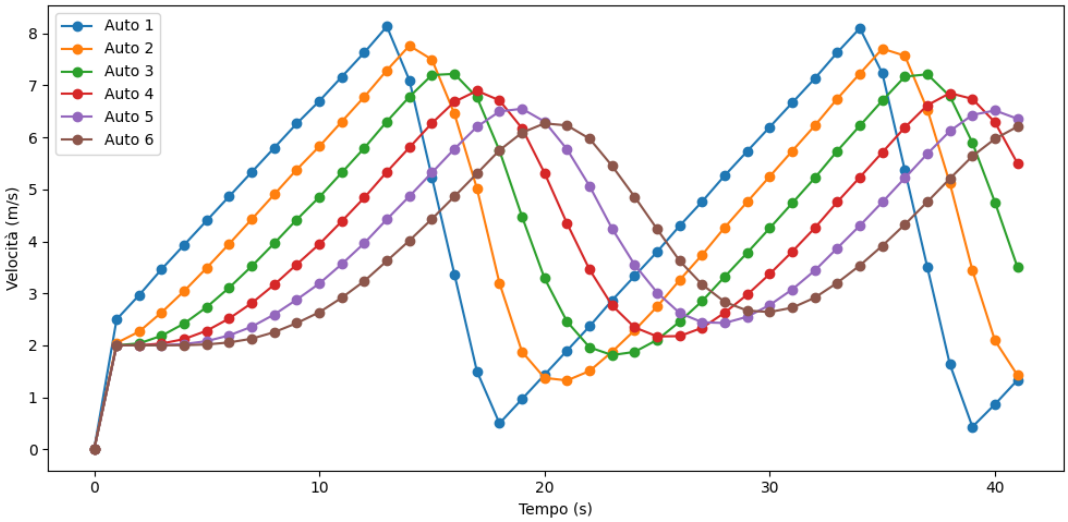

Figura 40: Velocità con τ a 0.01.

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.7	0.2	0.7

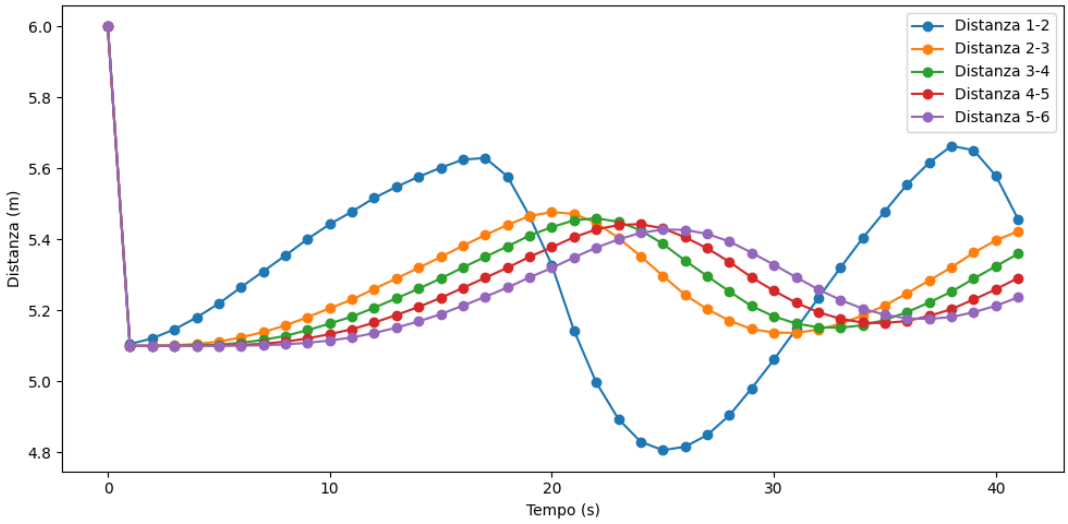

Figura 41: Distanze con τ a 0.7.

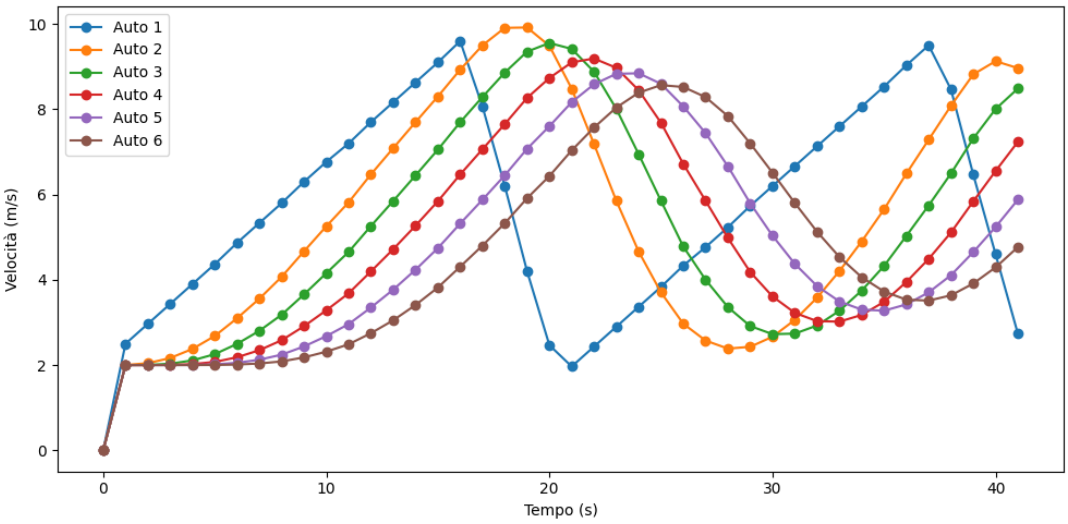

Figura 42: Velocità con τ a 0.7.

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.5	2.0	0.2	0.7

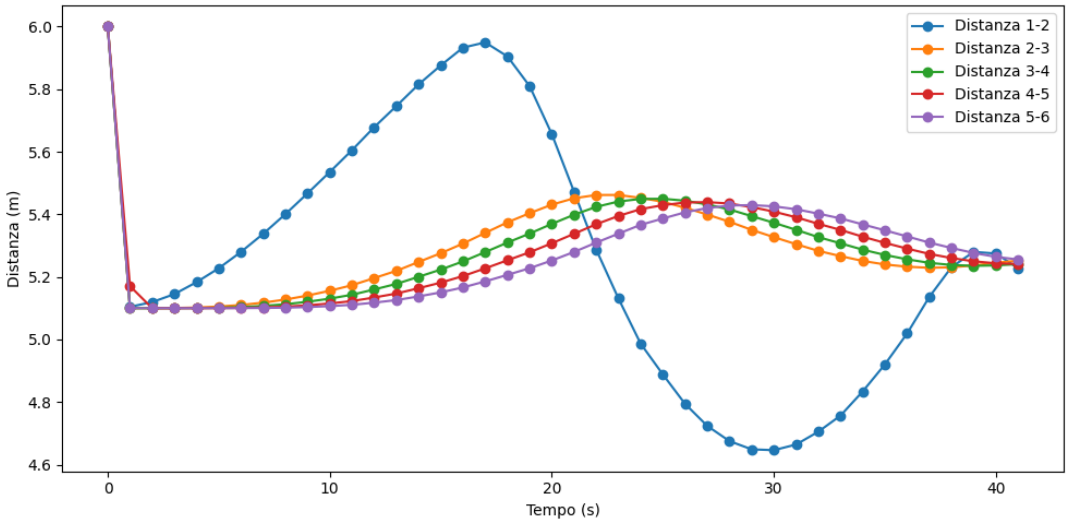

Figura 43: Distanze con τ a 2.0.

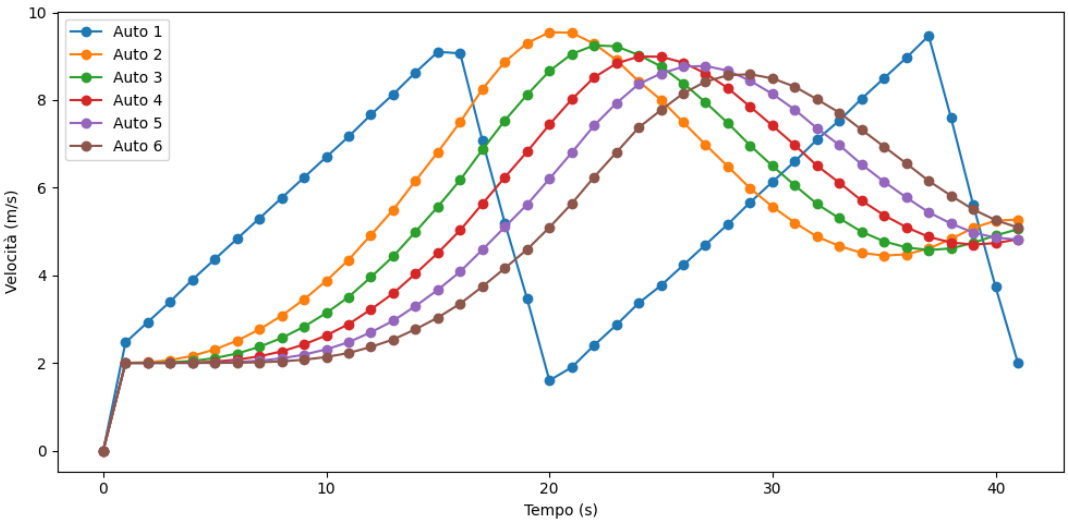

Figura 44: Velocità con τ a 2.0.

5.1.7 Velocità con parametri di default

Velocità $t1$	Velocità $t2$	Velocità $t3$	Velocità $t4$	Velocità $t5$
$0\,m/s$	$0\,m/s$	$0\,m/s$	$0\,m/s$	$0\,m/s$

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

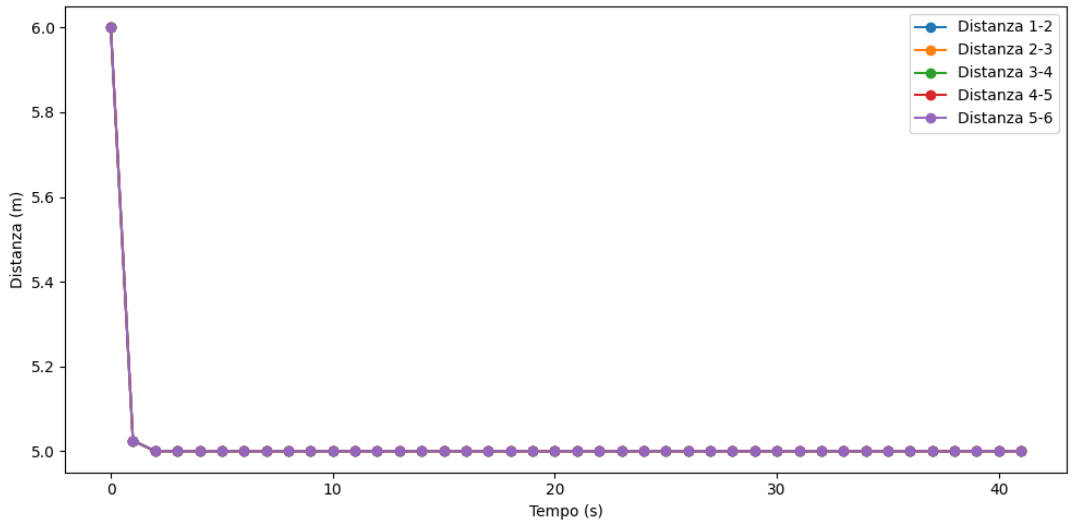

Figura 45: Distanze con primo veicolo fermo.

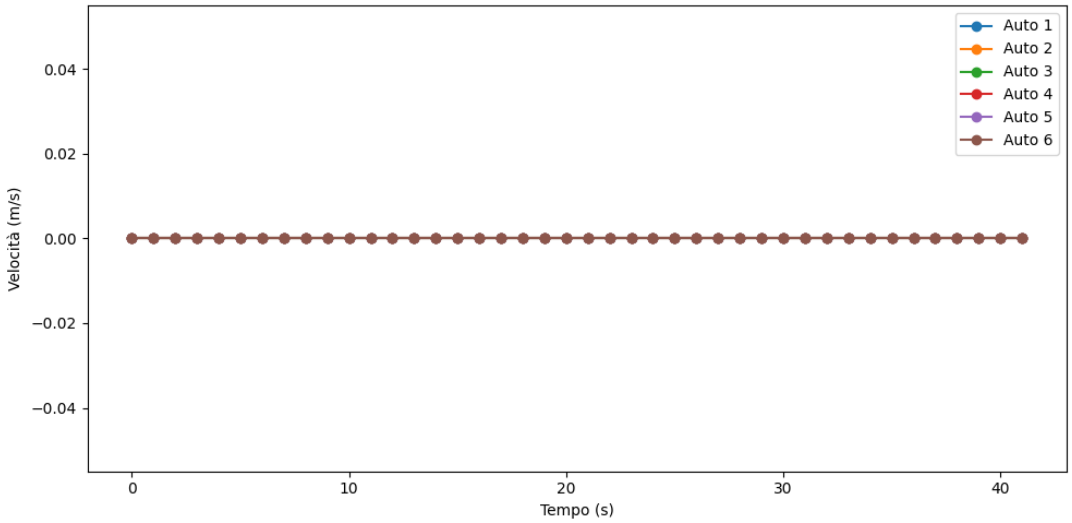

Figura 46: Velocità con primo veicolo fermo.

Velocità $t1$	Velocità $t2$	Velocità $t3$	Velocità $t4$	Velocità $t5$
$10\,m/s$	$10\,m/s$	$10\,m/s$	$10\,m/s$	$10\,m/s$

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

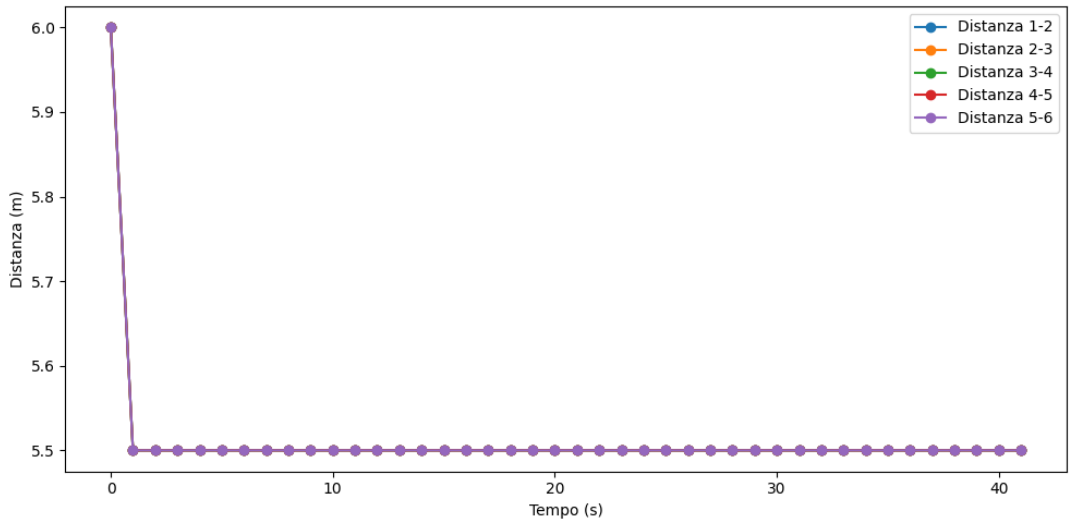

Figura 47: Distanze con velocità costante a $10\,m/s$.

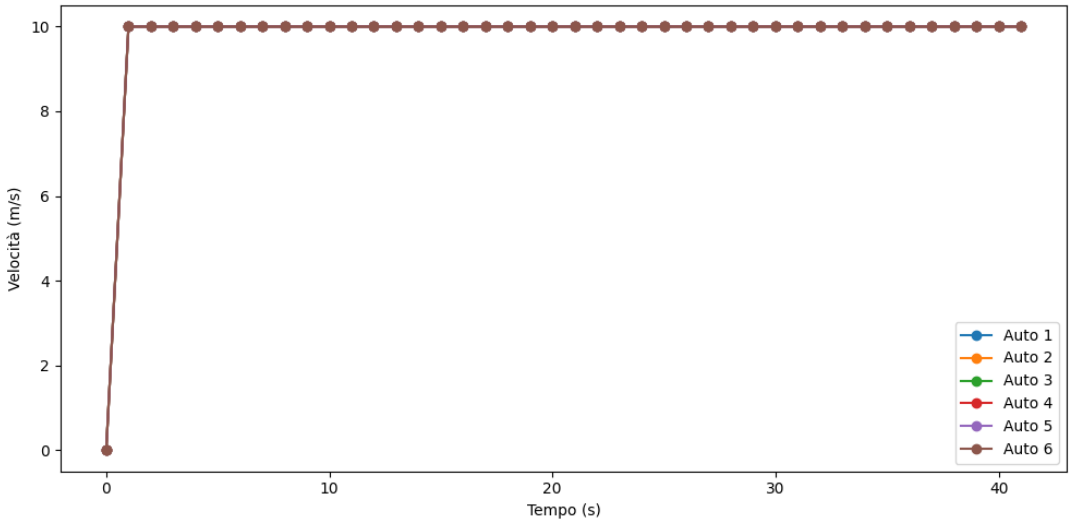

Figura 48: Velocità con velocità costante a $10\,m/s$.

Velocità $t1$	Velocità $t2$	Velocità $t3$	Velocità $t4$	Velocità $t5$
$35\,m/s$	$35\,m/s$	$35\,m/s$	$35\,m/s$	$35\,m/s$

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

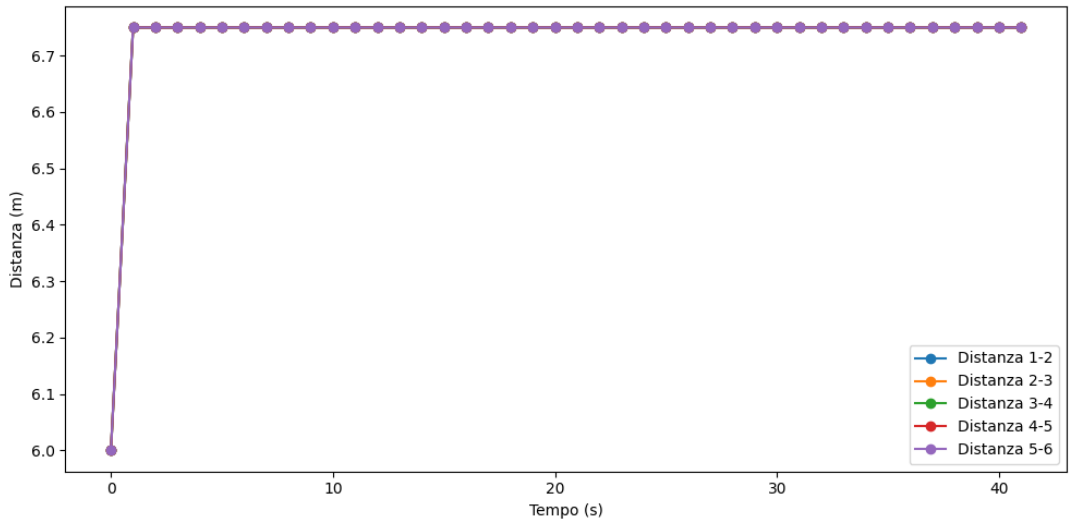

Figura 49: Distanze con velocità costante a $35\,m/s$.

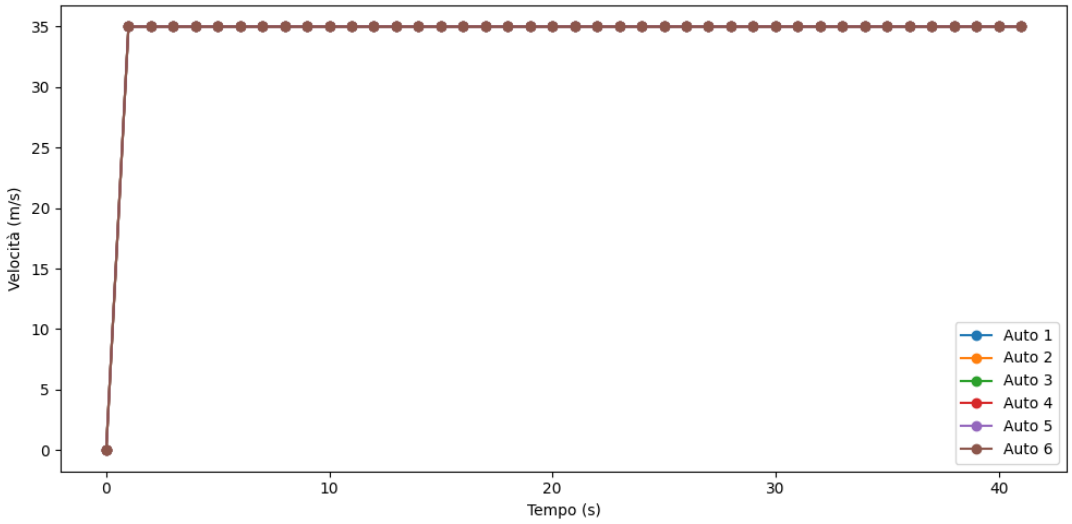

Figura 50: Velocità con velocità costante a $35\,m/s$.

Velocità $t1$	Velocità $t2$	Velocità $t3$	Velocità $t4$	Velocità $t5$
$20\,m/s$	$0\,m/s$	$20\,m/s$	$0\,m/s$	$20\,m/s$

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

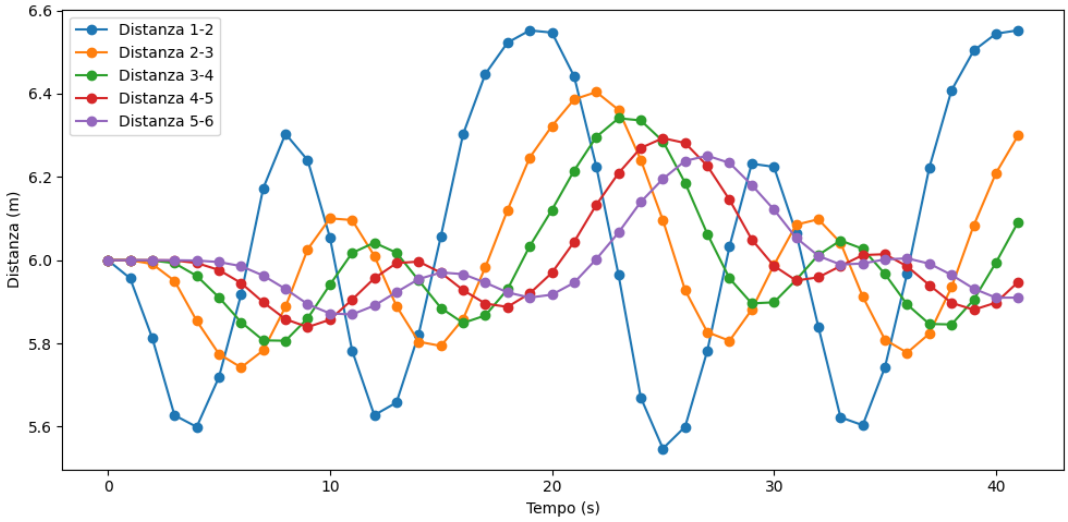

Figura 51: Distanze con velocità variabile tra $20\,m/s$ e $0\,m/s$.

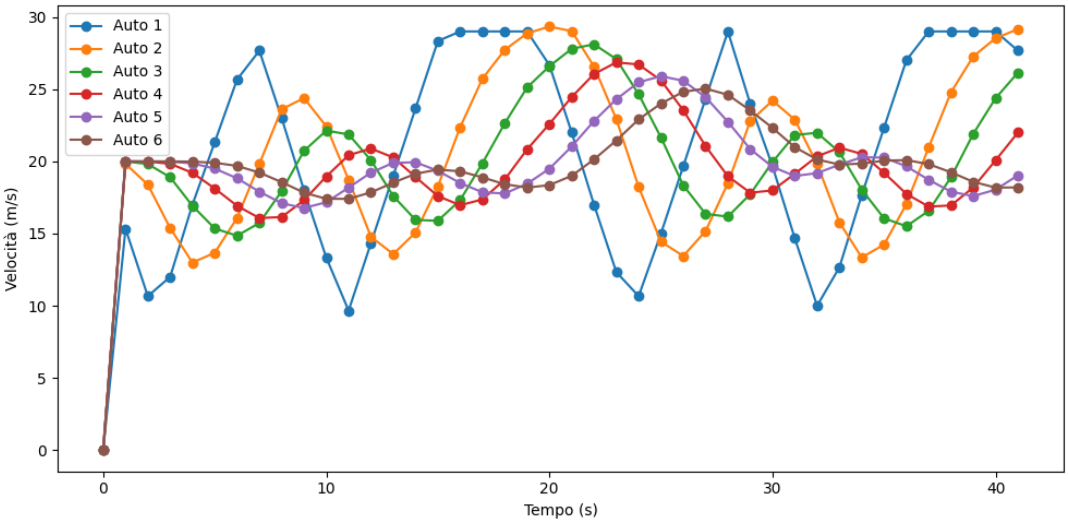

Figura 52: Velocità con velocità variabile tra $20\,m/s$ e $0\,m/s$.

Velocità $t1$	Velocità $t2$	Velocità $t3$	Velocità $t4$	Velocità $t5$
$35\,m/s$	$35\,m/s$	$0\,m/s$	$35\,m/s$	$35\,m/s$

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

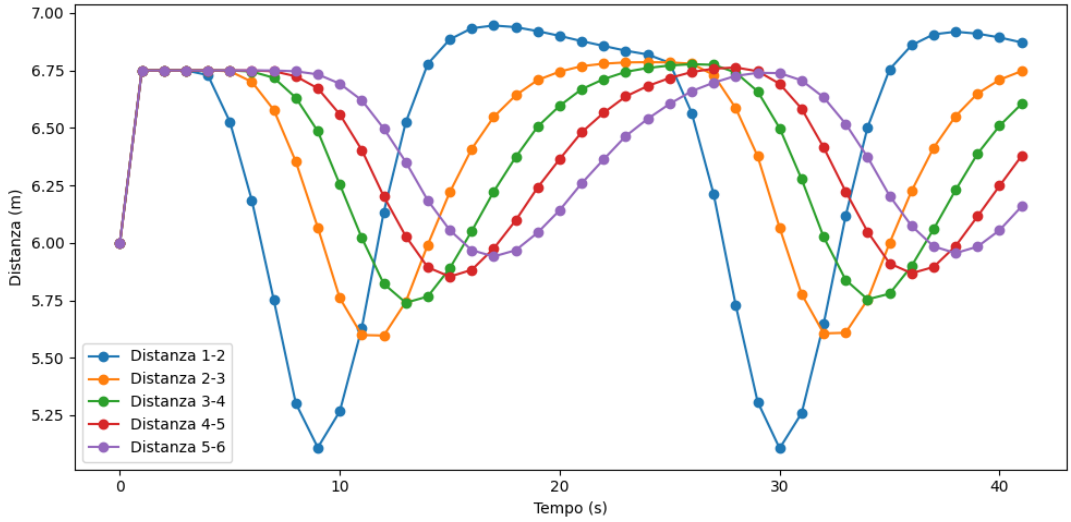

Figura 53: Distanze con velocità variabile tra $35\,m/s$ e $0\,m/s$.

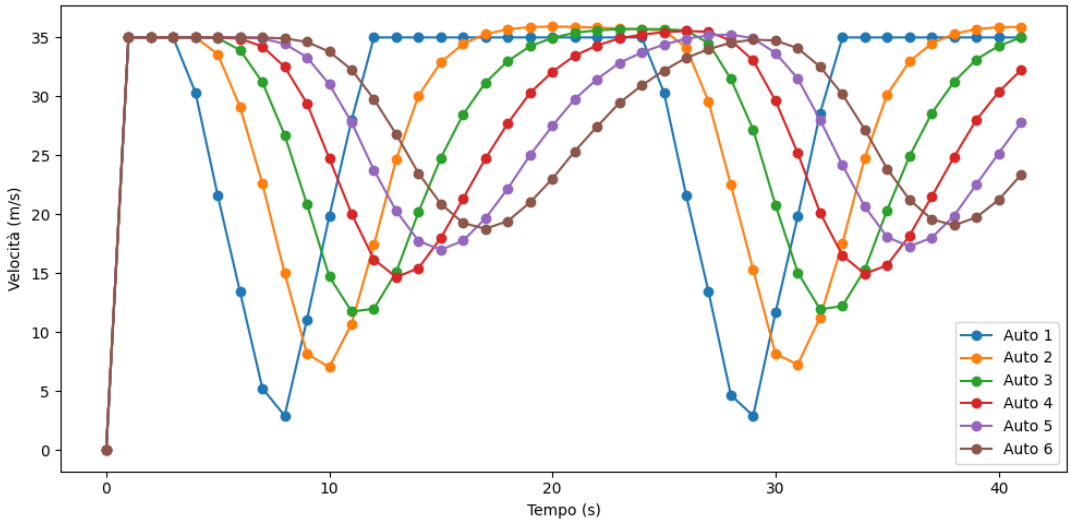

Figura 54: Velocità con velocità variabile tra $35\,m/s$ e $0\,m/s$.

5.1.8 Velocità con parametri diversi da quelli di default

Velocità $t1$	Velocità $t2$	Velocità $t3$	Velocità $t4$	Velocità $t5$
$15\,m/s$	$15\,m/s$	$15\,m/s$	$15\,m/s$	$15\,m/s$

N° auto	Distanza Iniziale	Distanza Target	Ritardo
10	$6.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.1	0.1	0.2	0.7

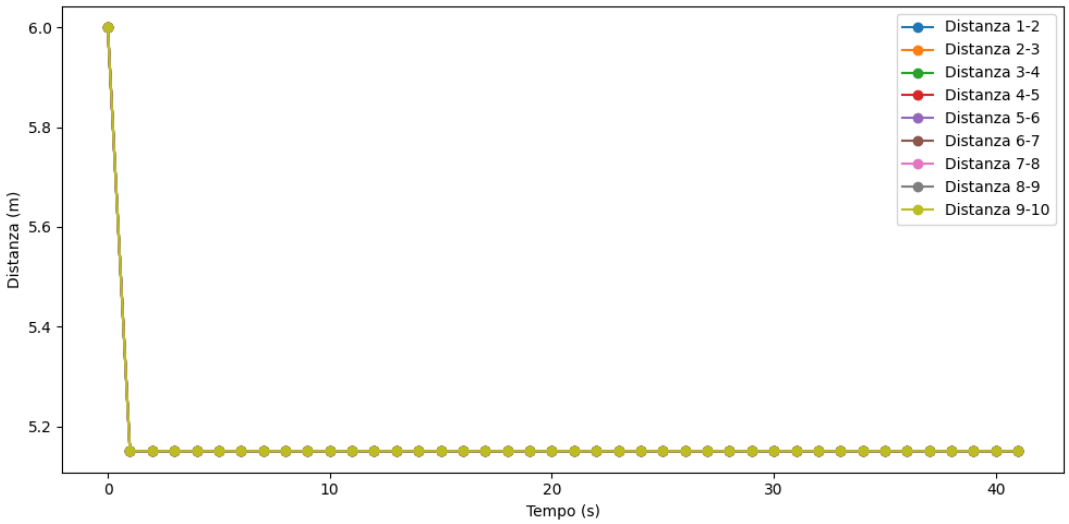

Figura 55: Distanze con velocità costante a $15\,m/s$.

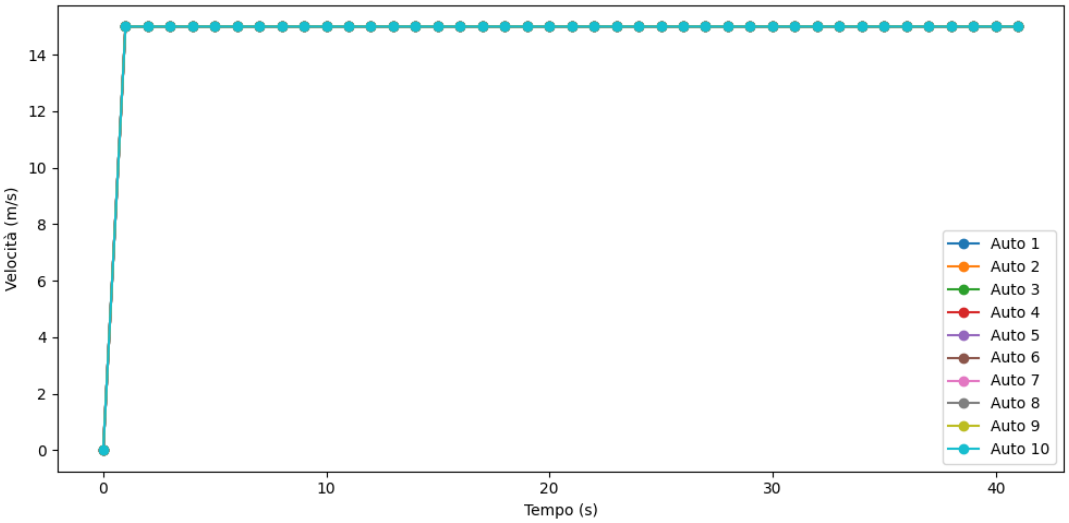

Figura 56: Velocità con velocità costante a $15\,m/s$.

Velocità $t1$	Velocità $t2$	Velocità $t3$	Velocità $t4$	Velocità $t5$
$25\,m/s$	$25\,m/s$	$25\,m/s$	$25\,m/s$	$25\,m/s$

N° auto	Distanza Iniziale	Distanza Target	Ritardo
10	$3.0m$	$2.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.1	0.1	0.2	0.7

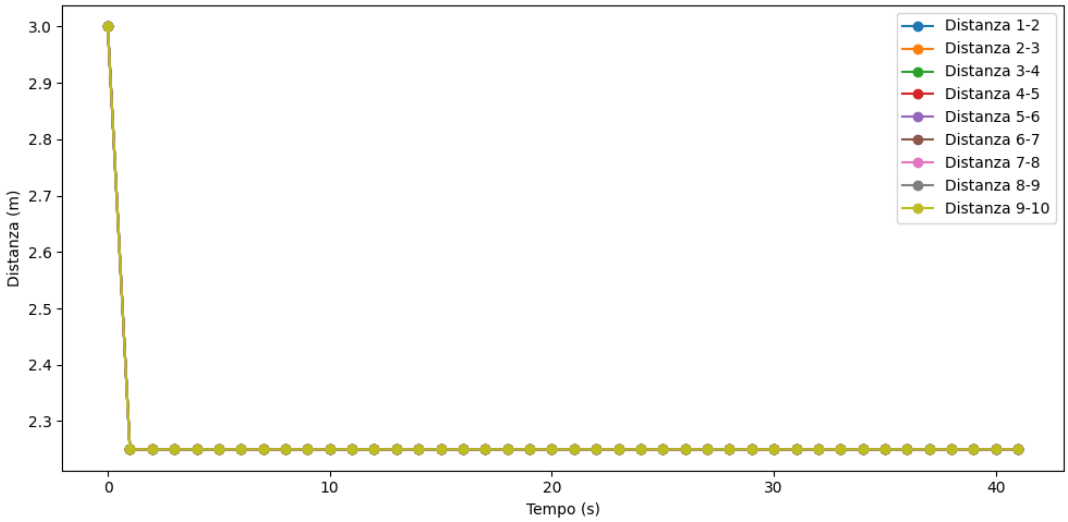

Figura 57: Distanze con velocità costante a $25\,m/s$.

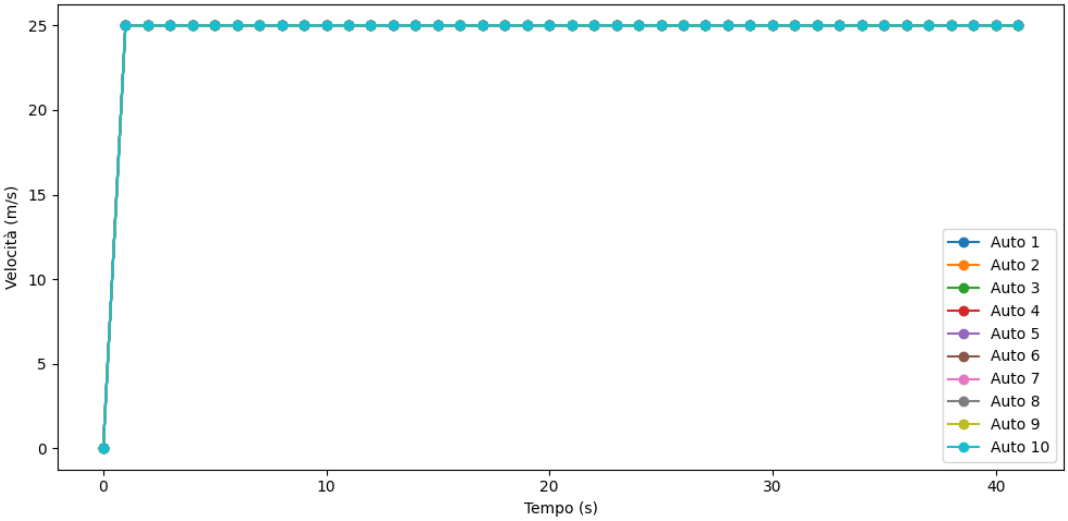

Figura 58: Velocità con velocità costante a $25\,m/s$.

Velocità $t1$	Velocità $t2$	Velocità $t3$	Velocità $t4$	Velocità $t5$
$25\,m/s$	$25\,m/s$	$15\,m/s$	$25\,m/s$	$25\,m/s$

N° auto	Distanza Iniziale	Distanza Target	Ritardo
10	$3.0m$	$5.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.1	0.1	0.2	0.7

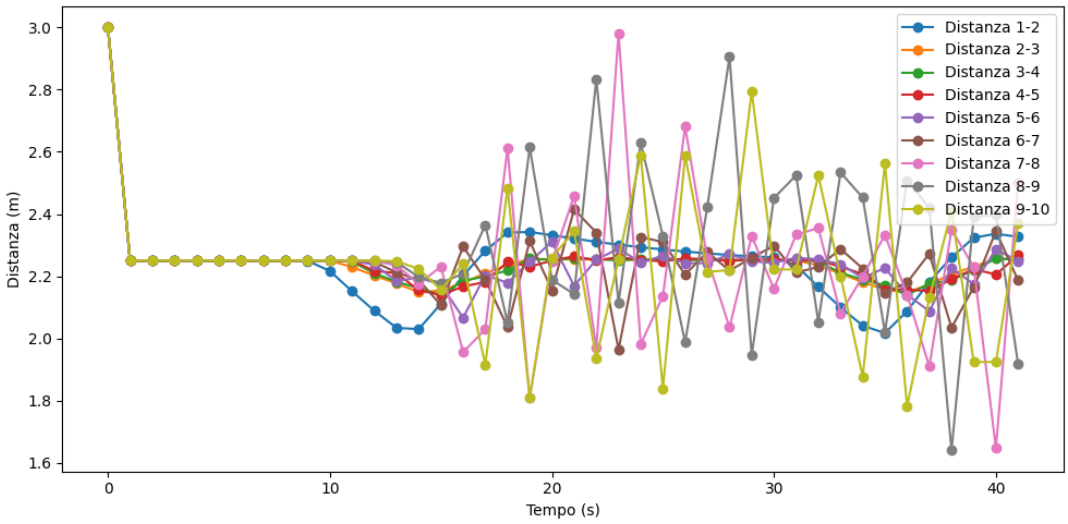

Figura 59: Distanze con velocità variabile tra $15\,m/s$ e $25\,m/s$.

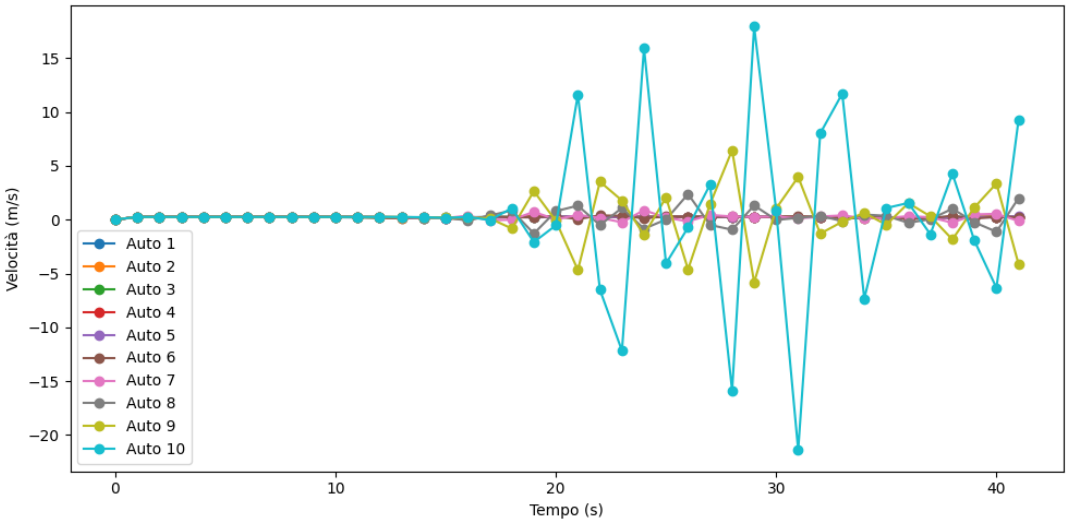

Figura 60: Velocità con velocità variabile tra $15\,m/s$ e $25\,m/s$.

5.2 Modello instabile

5.2.1 Ritardo di comunicazione

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$1.5s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

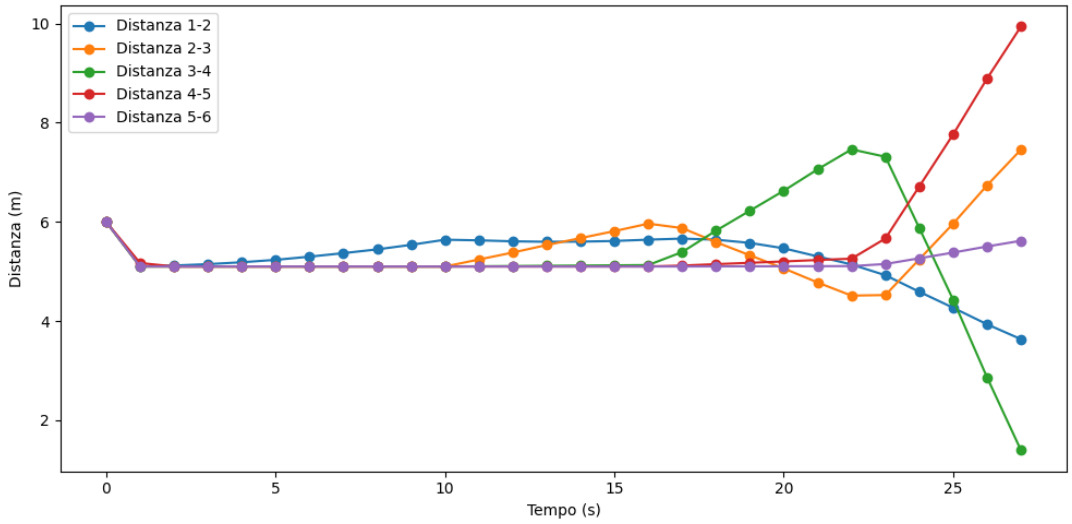

Figura 61: Distanze con ritardo a $1.5s$.

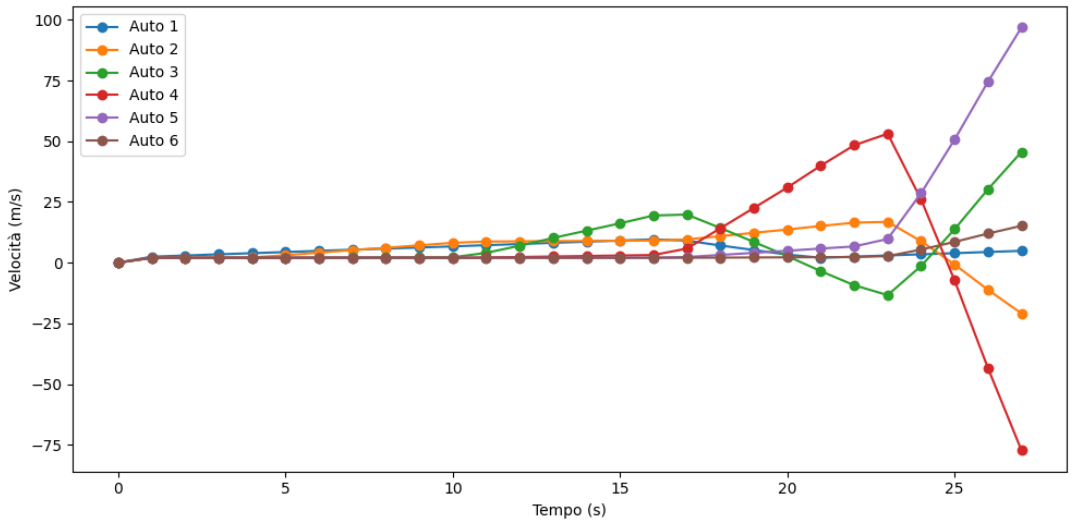

Figura 62: Velocità con ritardo a $1.5s$.

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$6.0m$	$5.0m$	$2s$
$Time\ Headway$	τ	K_p	K_d
0.5	0.1	0.2	0.7

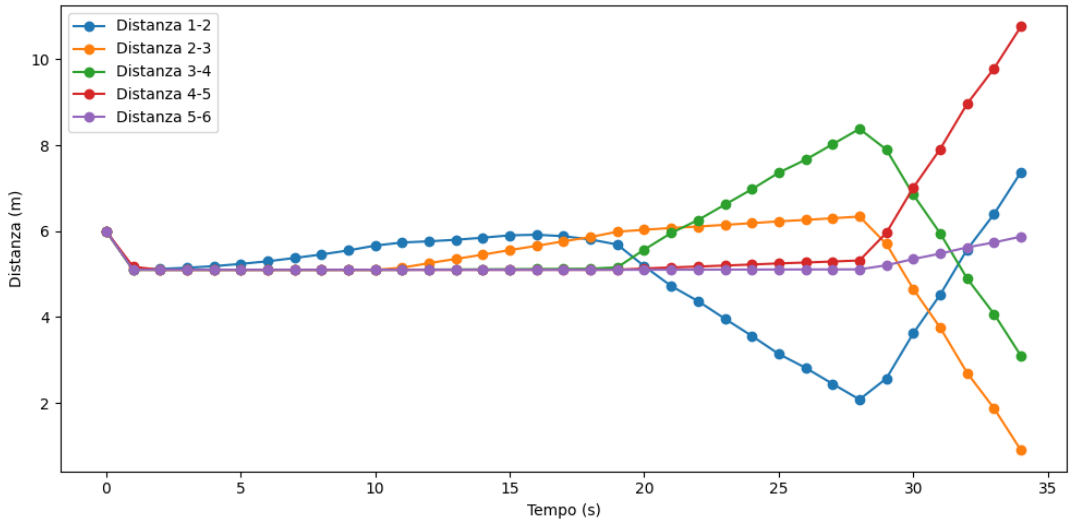

Figura 63: Distanze con ritardo a $2s$.

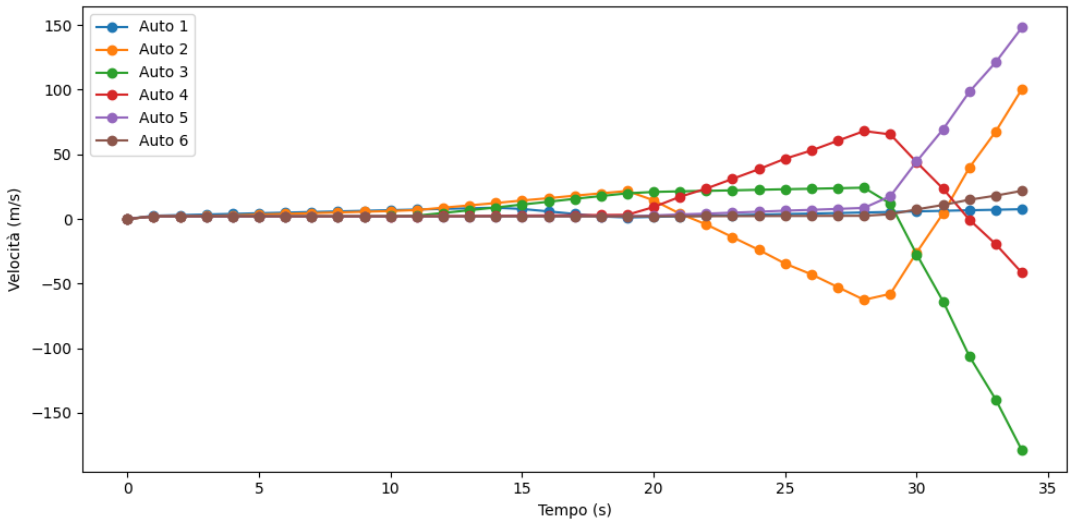

Figura 64: Velocità con ritardo a $2s$.

5.2.2 Velocità con parametri diversi da quelli di default

Velocità $t1$	Velocità $t2$	Velocità $t3$	Velocità $t4$	Velocità $t5$
$35\,m/s$	$20\,m/s$	$0\,m/s$	$15\,m/s$	$35\,m/s$

N° auto	Distanza Iniziale	Distanza Target	Ritardo
6	$3.0m$	$2.0m$	$0.2s$
$Time\ Headway$	τ	K_p	K_d
0.05	0.1	0.2	0.7

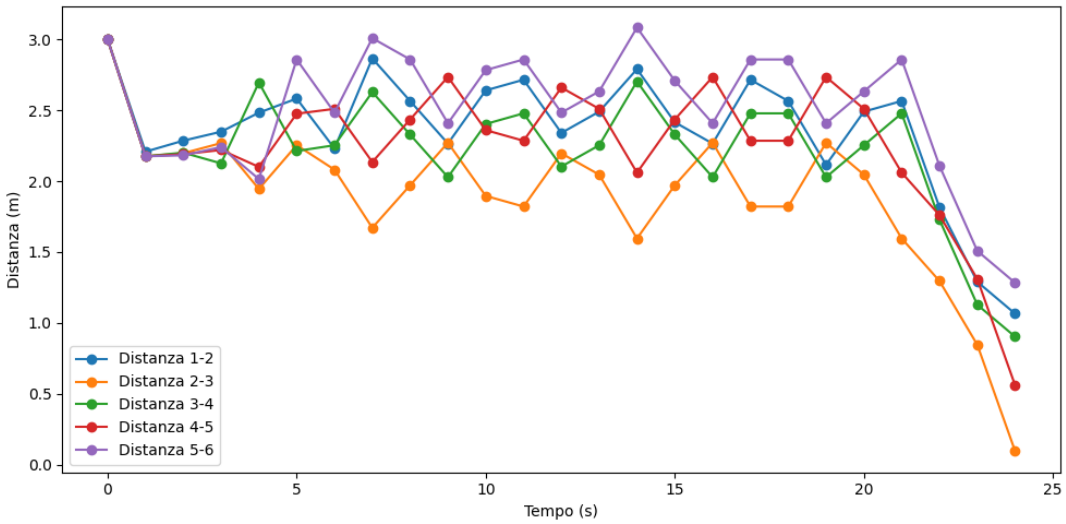

Figura 65: Distanze con velocità variabile tra $0\,m/s$ e $35\,m/s$.

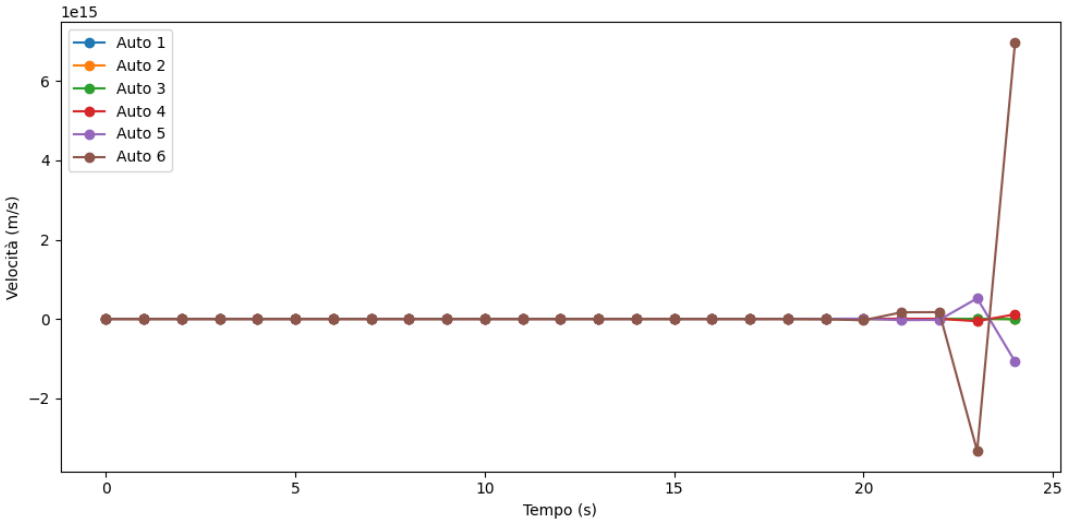

Figura 66: Velocità con velocità variabile tra $0\,m/s$ e $35\,m/s$.

Velocità $t1$	Velocità $t2$	Velocità $t3$	Velocità $t4$	Velocità $t5$
$35\,m/s$	$20\,m/s$	$0\,m/s$	$15\,m/s$	$35\,m/s$

N° auto	Distanza Iniziale	Distanza Target	Ritardo
7	$4.5m$	$3.5m$	$1.0s$
$Time\ Headway$	τ	K_p	K_d
0.3	0.1	0.2	0.7

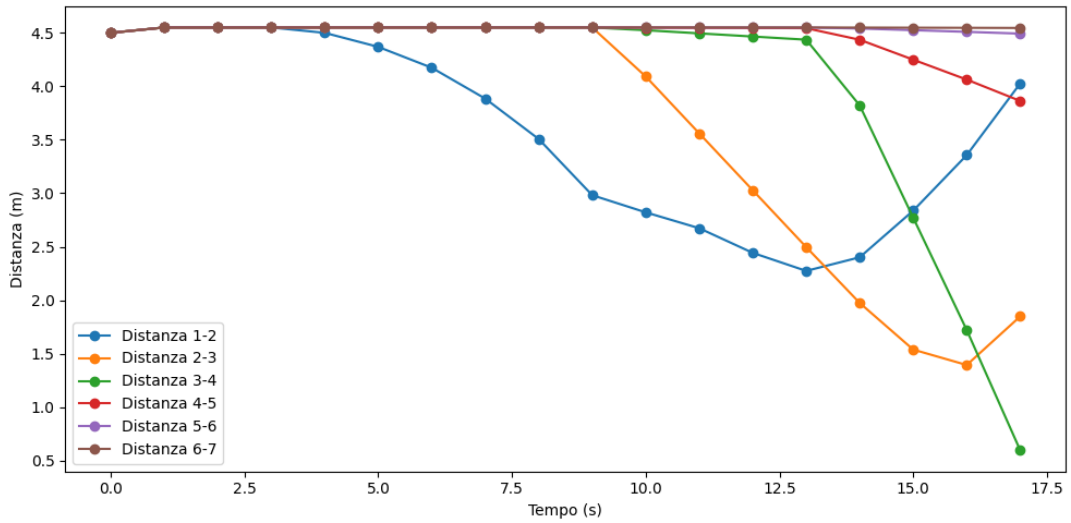

Figura 67: Distanze con velocità variabile tra $0\,m/s$ e $35\,m/s$.

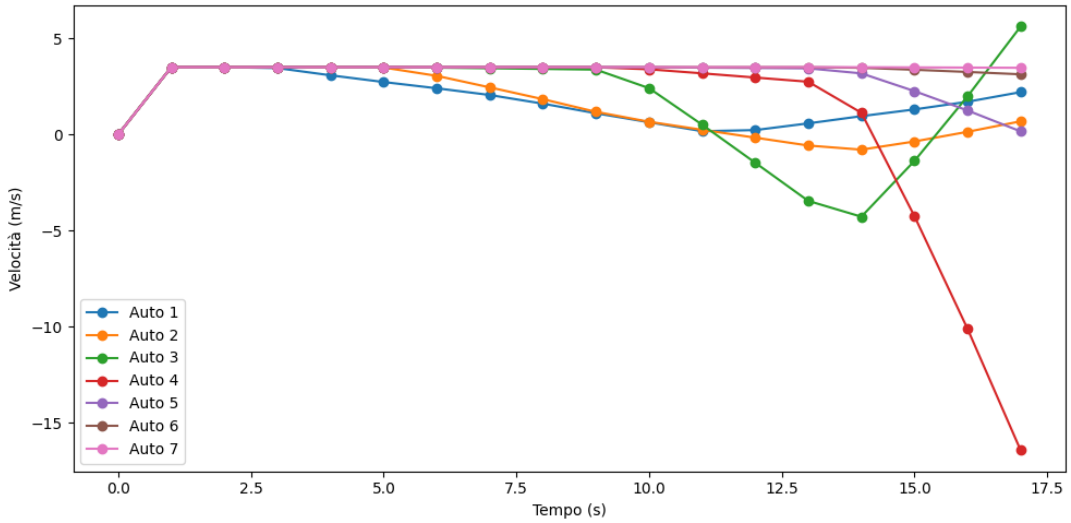

Figura 68: Velocità con velocità variabile tra $0\,m/s$ e $35\,m/s$.

6 Conclusioni

In allegato lasciamo la presentazione [DM24] che riassume tutto il progetto in formato di diapositive, per una consultazione rapida.

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.

[Qua16] Autoscuola Quattroruote AQ. *Distanza di Sicurezza*. Immagine recuperata da autoscuola-quattroruote.com[17]. Mag. 2016.

[Onl21] Patentino Online PO. *Distanza di Sicurezza*. Immagini recuperate da patentinoonline.it[18]. 2021.

[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 qui[19]. Firenze, Italia, apr. 2024.

[17] https://www.autoscuola-quattroruote.com/portfolio/distanza-di-sicurezza/

[18] https://www.patentinoonline.it/distanza-di-sicurezza

[19] https://docs.google.com/viewer?url=https://github.com/albbus-stack/platooning-visualization/blob/main/doc/Presentazione_Platooning.pdf?raw=true

Elenco delle figure

Ringraziamenti