Frontend per la simulazione di platooning di veicoli

Alberto Del Buono Paolini - Federico Marra

Relatore: Giorgio Battistelli

Corso di Laurea Triennale in Ingegneria Informatica
Università degli Studi di Firenze

Aprile 2024

Tabella dei contenuti

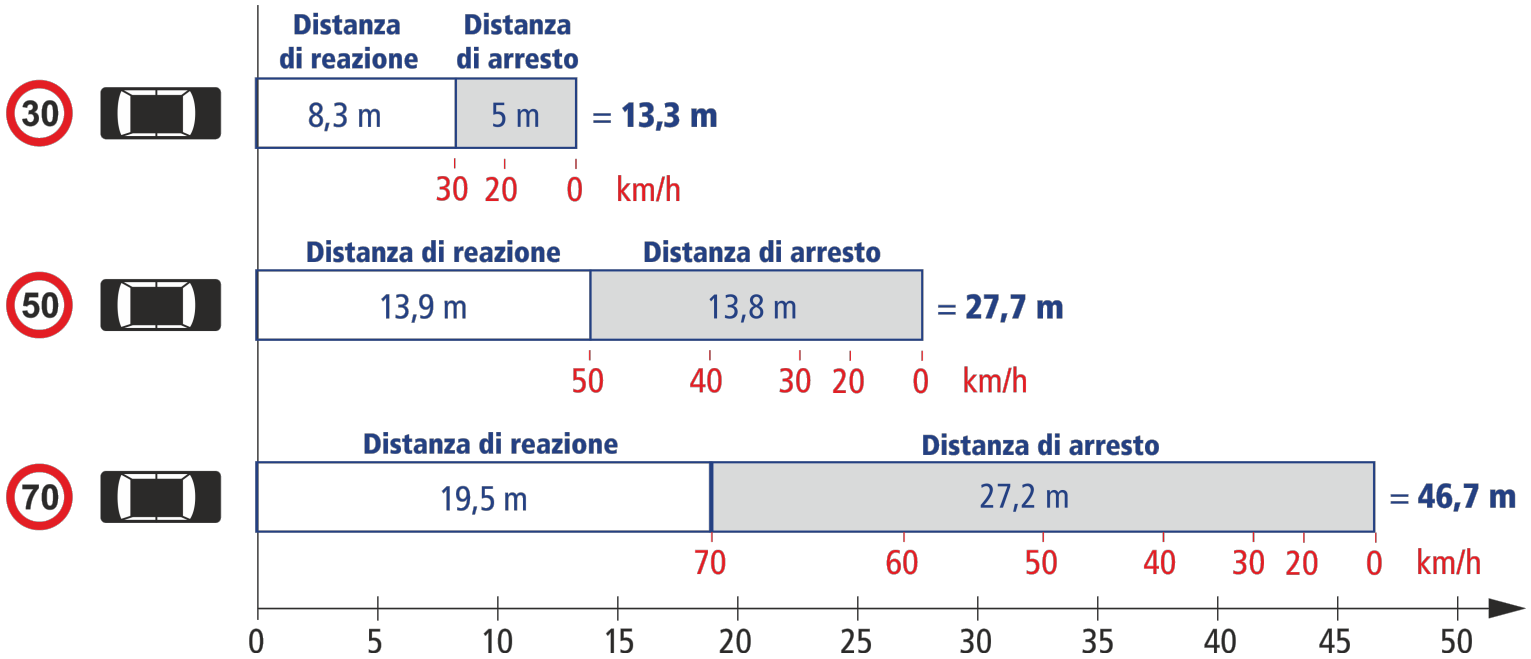
Cos'è il platooning

Il ***platooning*** o plotonamento è un sistema avanzato di guida autonoma, detto anche **CACC** (*Cooperative Adaptive Cruise Control*) in cui veicoli si muovono in modo coordinato e autonomo, formando un convoglio o plotone di veicoli.

I maggiori **vantaggi** sono:

- Riduzione della distanza tra i veicoli.
- Ottimizzazione del flusso del traffico.
- Aumento della sicurezza.
- Riduzione consumo di carburante ed emissioni.

Distanza di sicurezza



Un essere umano ha mediamente un **tempo di reazione** di $1\,s$ che determina uno spazio di reazione proporzionale alla velocità.
Automatizzare la reazione permette di **diminuire** questa distanza.

Plotone di veicoli

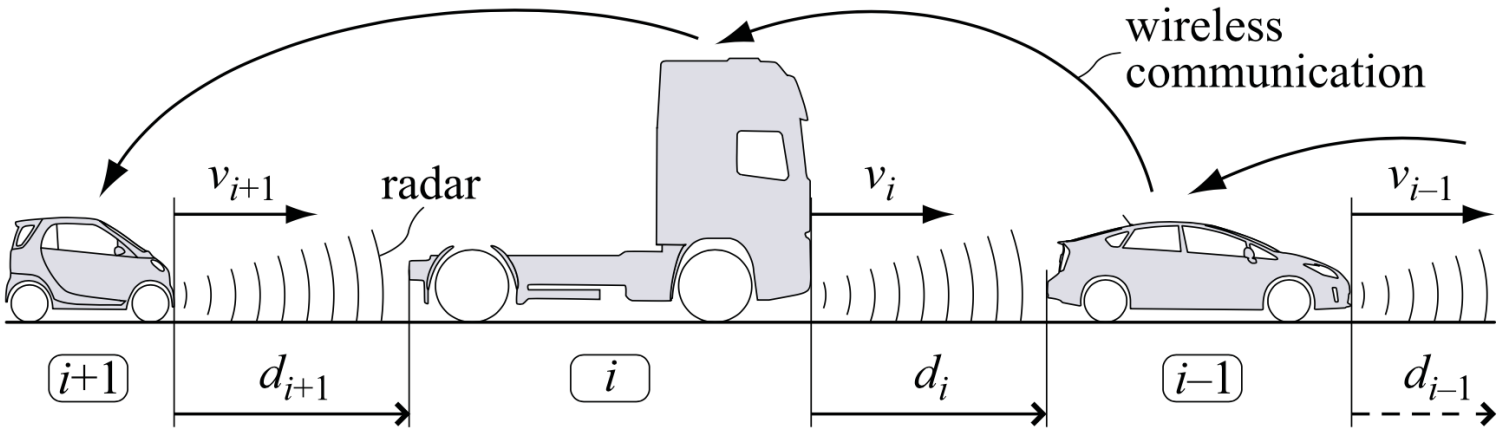
Il sistema si pone di mantenere una **distanza costante** prestabilita tra i veicoli, minore della distanza di sicurezza, le vetture successive si adattano quindi in modo coordinato alle variazioni di velocità e direzione del veicolo di testa.

I veicoli usano sensori per **regolare automaticamente** la velocità e la distanza all'interno del convoglio:

- Telecamere
- Radar
- Lidar

Tabella dei contenuti

Controllo di crociera adattivo e cooperativo (CACC)



Il primo veicolo è detto **pilota** ed è quello che non è soggetto a controllo, detta lui la velocità del plotone.
A catena ogni veicolo **segue** quello che lo precede (i segue $i-1$).

Legge di controllo

La legge di controllo è basato su un **PID** (proporzionale integrale derivativo) che agisce contemporaneamente su ogni veicolo del plotone tranne quello pilota.

La **legge di controllo** è ottenuta dal sistema:

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

Parametri di controllo

- Time Headway (h)
- Tau (τ)
- Kp (k_p)
- Kd (k_d)
- Delay

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

N° auto	6
Distanza obiettivo	$5m$
Delay	$0.2s$

Sistema fisico

Per la **discretizzazione** delle equazioni la frequenza di campionamento è di $30\,Hz$ corrispondenti a $30\,fps$.

Usiamo il **metodo di Eulero**:

$$x_i(t) = x_i(t-1) + \frac{\Delta x_i(t-1)}{F_s}$$

Definiamo $d_i(t)$ come la distanza fra i veicoli i e $i-1$:

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

Stabilità di stringa

Legata al concetto di stabilità di Lyapunov, la **stabilità di stringa** è prettamente legata alle distanze lungo un unico asse. Nella nostra simulazione le auto si muovono unicamente seguendo le ascisse.

Consideriamo il sistema stabile se:

$$\lim_{t\to\infty} e_i(t) = 0 \qquad i \in S_m$$

Dunque le distanze tra i veicoli rimangono **consistenti** nel tempo.

Tabella dei contenuti

Stack delle tecnologie

Stack

- **React**: Libreria popolare di rendering UI e di gestione dello stato locale.
- **Next.js**: Framework specializzato nel rendering lato server che offre prestazioni ottimali e una struttura di sviluppo solida.
- **p5.js**: Framework grafico utilizzato per la creazione del canvas di simulazione.
- **Chart.js**: Libreria per il rendering dei grafici per fornire una visualizzazione chiara delle informazioni.
- **ParaglideJS**: Libreria per la gestione e il mantenimento dell'internazionalizzazione.

React

Abbiamo scelto *React* come libreria di rendering per il front-end e per la gestione dello stato locale. E' stata usata l'API React per la gestione del contesto locale, questa include un ***provider*** e un ***hook*** (`useContext`) per consumarlo nei vari componenti sottostanti.

Contesto locale

```
// DataProvider.tsx
export const DataContext = createContext({
  graphData: [[]] as DataType[][],
})

// GraphSliver.tsx
const { graphData } = useContext(DataContext);
```

Next.js

Abbiamo usato le *Dynamic Routes* per creare dinamicamente percorsi nel server per ogni lingua, rendendo facile l'aggiunta di nuove lingue. Oltretutto tutte le pagine del sito sono mantenute in ***cache*** sul server, velocizzando i tempi di caricamento.

Routing dinamico

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

  return <HomePage />;
};
```

p5.js

L'utilizzo di *p5.js* ci consente di ottenere una **simulazione** ad alto frame rate, garantendo una rappresentazione fluida del platooning da cui poi campioniamo i dati per generare i vari grafici.
Questa libreria si interfaccia con *React* usando un **wrapper** per essere aggiornata ogni volta che lo stato locale dell'applicazione cambia.

Canvas di simulazione

```
<NextReactP5Wrapper
  sketch={sketch}
  // Altre impostazioni per la simulazione ...
  resetCanvas={resetCanvas}
  togglePlay={togglePlay}
/>
```

Chart.js

Chart.js ci permette di integrare facilmente grafici interattivi nella UI dell'applicazione, includendo anche animazioni all'aggiunta di nuovi dati. Usiamo un **grafico interattivo** che modifica lo stato *React* per la selezione delle velocità della macchina iniziale del convoglio.

Esempio di grafico

```
<Line
 data={{
  labels: graphData[velocityChartIndex].map((d) => d.time),
  datasets: [{
    data: graphData[velocityChartIndex].map((d) => d.
        velocity),
   },],}}
/>
```

ParaglideJS

Usiamo paraglide-js per gestire i vari file `json` che contengono le chiavi per le **traduzioni**. Questa libreria offre anche la gestione per lingua selezionata dall'utente sulla *frontend*, esponendo nello stato locale solo le traduzioni corrispondenti alla selezione.

Esempio di configurazione

```
// project.inlang.json
{
  "sourceLanguageTag": "en",
  "languageTags": [ "en", "it", ... ],
  "plugin.inlang.messageFormat": {
    "pathPattern": "./translations/{languageTag}.json"
  }
}
```

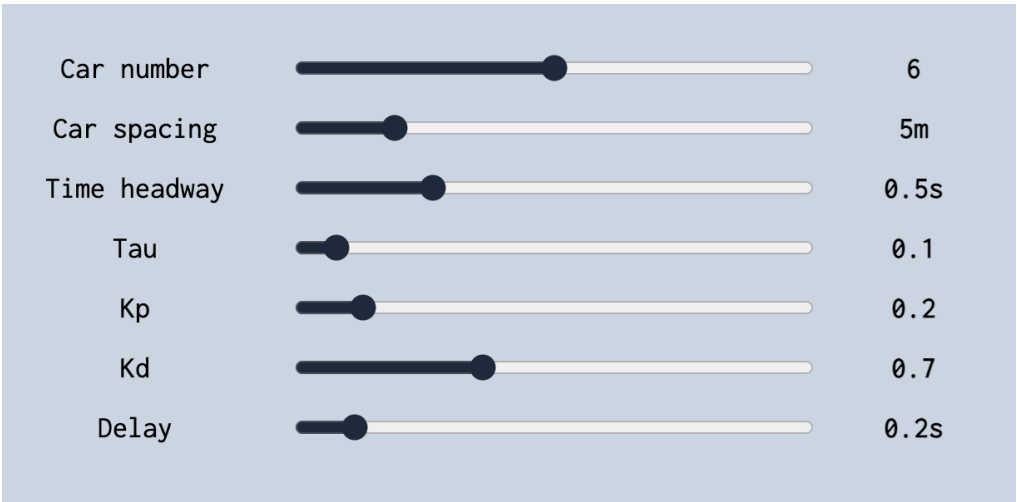
Pubblicazione e *CI/CD*

Questa applicazione può essere **pubblicata**, come qualsiasi altro progetto *Next.js*, tramite *Netlify*, *AWS Amplify* o *Vercel*.

Abbiamo optato per *Vercel* dato che offre **integrazione/distribuzione continua** (*CI/CD*) per il rilascio dell'app ogni volta che viene eseguito un *commit* o una *pull request* sul branch principale della repository Github, offrendo anche *deployment* di anteprima non disponibili al pubblico.

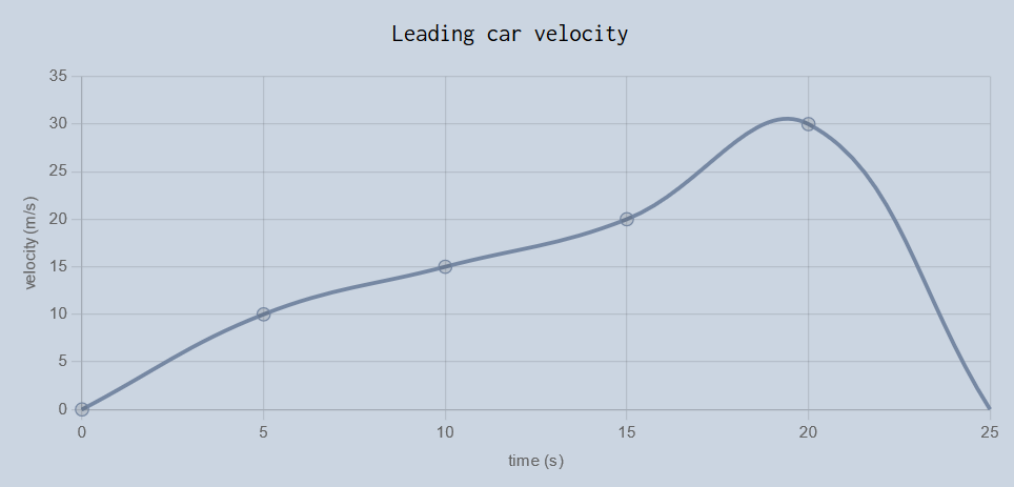
Tabella dei contenuti

Parametri di simulazione interattivi



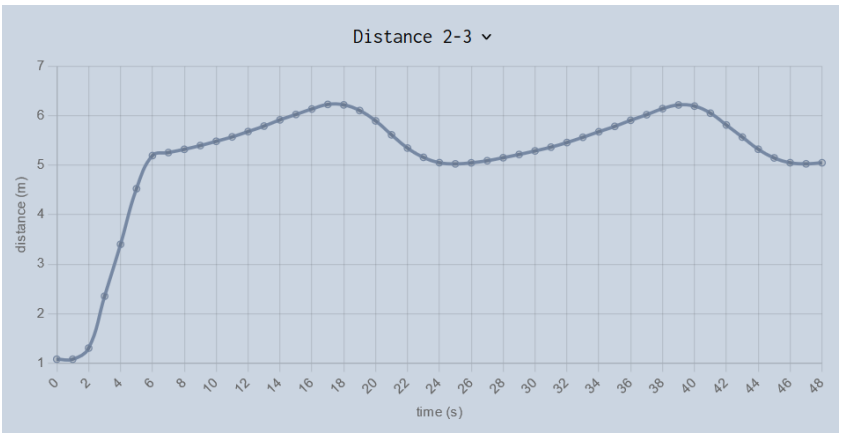
- Regolazione dinamica del **numero di veicoli** e della **distanza obiettivo**.
- Impostazioni per tutti i parametri della simulazione: *Tau*, *Kp*, *Kd*, *Time headway* e *Delay*

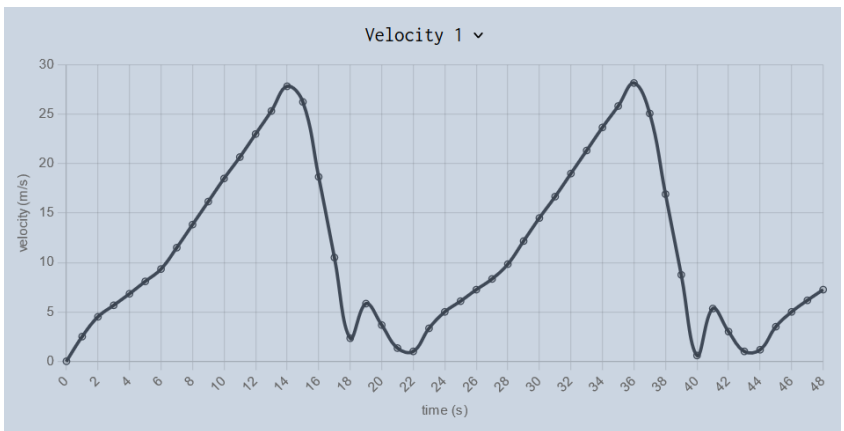
Parametri di simulazione interattivi



- Interfaccia per la regolazione della **velocità del primo veicolo** del convoglio, l'utente può disegnare il grafico per punti che si ripetono periodicamente.

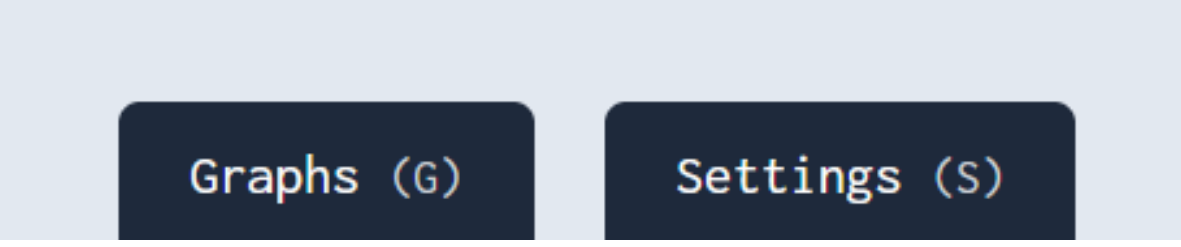
Visualizzazione dei dati campionati





- L'utente può navigare i **grafici** contenenti i dati campionati dalla simulazione su:
  - **Distanza** tra due veicoli
  - **Velocità** di un veicolo

Scorciatoie da tastiera



- L'interfaccia è navigabile anche usando delle **scorciatoie** da tastiera:

`G`: attiva/disattiva la tab dei grafici
`S`: attiva/disattiva la tab delle impostazioni
`SPAZIO`: riproduce/mette in pausa la simulazione
`R`: ferma e reimposta la simulazione

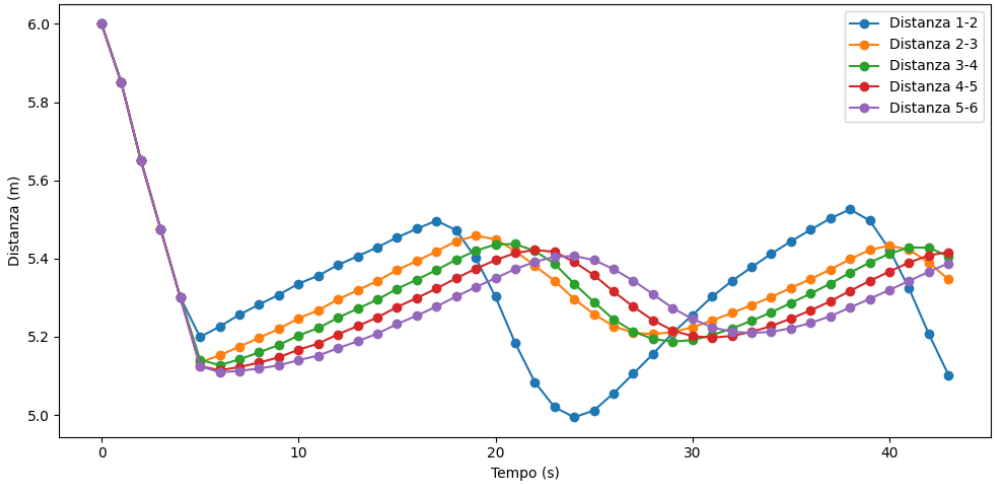
Download dei dati campionati

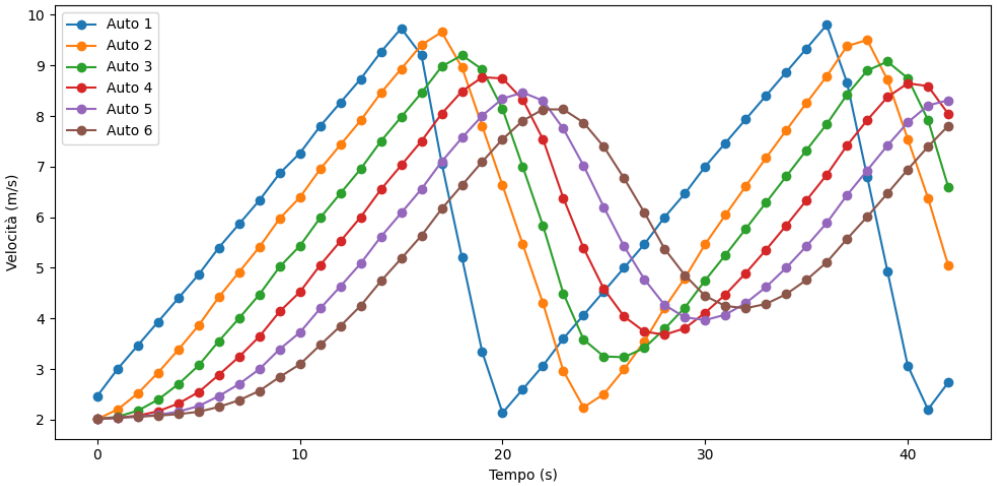
- L'utente può anche **scaricare i dati** come `csv` col seguente formato:

```
car,
time(s),
distance(m),
velocity(m/s)
```

Tabella dei contenuti

Modello stabile di esempio



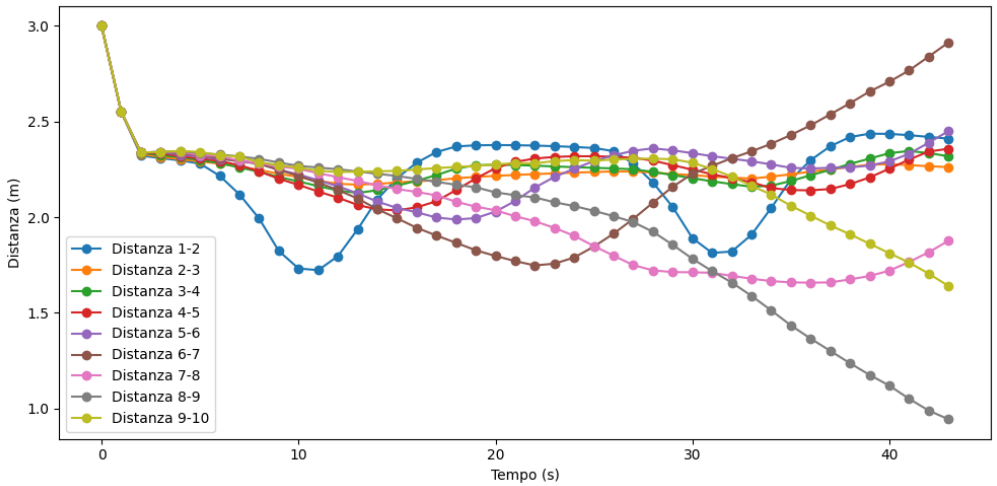


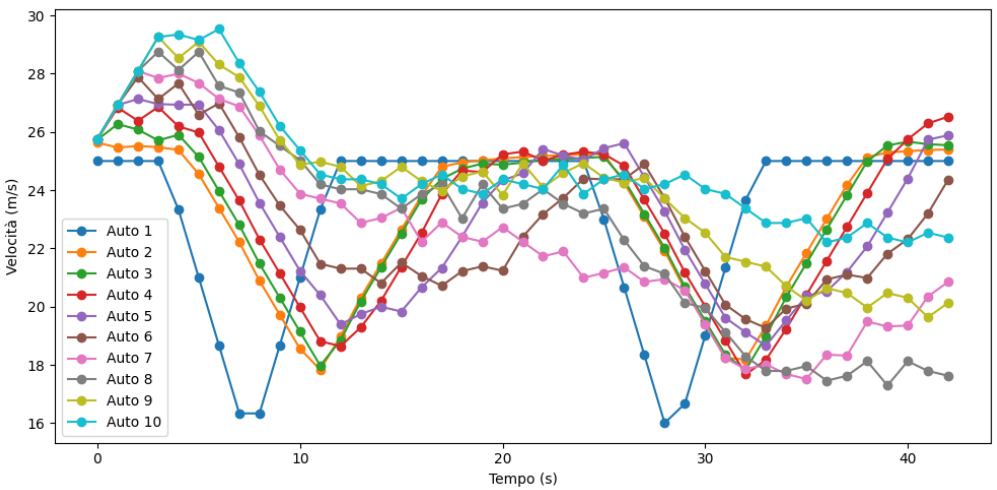
N° auto	Distanza Iniziale	Distanza Obiettivo	Ritardo
6	$6.0m$	$5.0m$	$0.2s$
Time Headway (h)	Tau (τ)	k_p	k_d
$0.5s$	0.1	0.2	0.7

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

Questa simulazione è stata usata come punto di partenza per poi valutare **singolarmente** l'impatto delle variazioni di ogni parametro.

Modello instabile di esempio





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
$25\,m/s$	$25\,m/s$	$15\,m/s$	$25\,m/s$	$25\,m/s$

Con tanti veicoli, tempo di separazione e distanza obiettivo ridotti, il modello non mantiene sempre la **stabilità**.

Conclusioni degli esperimenti

I parametri del modello che più influenzano la stabilità della simulazione sono il ***time headway*** (tempo di separazione tra i veicoli) e il ***delay*** (ritardo di comunicazione).

Oltre a questi, anche una variazione significativa della **velocità della prima vettura** può portare il sistema all'instabilità, essendo una perturbazione esterna al modello stesso.

Demo

platooning-simulation.vercel.app