Frontend per la simulazione di un problema di platooning

Alberto Del Buono Paolini - Federico Marra

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

Aprile 2024

Tabella dei contenuti

Cos'è il platooning

Il **platooning** è un sistema avanzato di guida autonoma in cui veicoli autonomi si muovono in modo coordinato e vicino tra loro, formando una sorta di *convoglio* o *traino*.

- Riduzione della distanza tra i veicoli per migliorare l'efficienza del trasporto.
- Coordinazione intelligente per evitare collisioni e ottimizzare il flusso del traffico.

Modello di controllo

Il modello di controllo nel contesto del platooning implica l'implementazione di algoritmi intelligenti che permettono ai veicoli di comunicare tra loro e di regolare la loro velocità in modo sincronizzato.

- Utilizzo di sensori avanzati per la percezione dell'ambiente.
- Algoritmi di controllo per mantenere la distanza e la velocità ottimali.

Slide 1 da aggiungere

—

- —
- —

Slide 2 da aggiungere

—

- —
- —

Slide 3 da aggiungere

—

- —
- —

Slide 4 da aggiungere

—

- —
- —

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.
- **Paraglide-js**: 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),
   },],}}
/>
```

Paraglide-js

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 front-end, 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*, cioè tramite *Netlify*, *AWS Amplify* o *Vercel*.

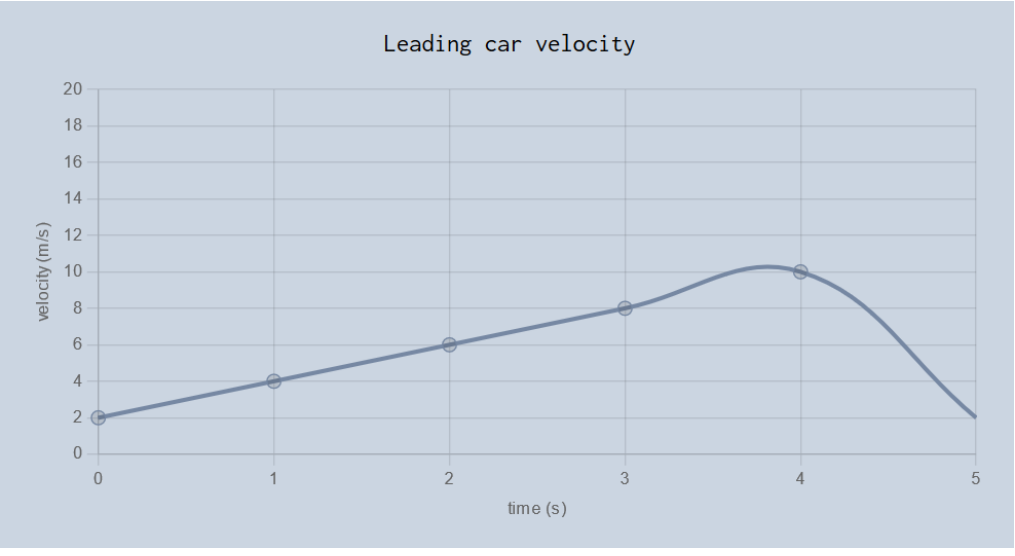
Abbiamo optato per *Vercel* dato che offre **integrazione/distribuzione continua** (*CI/CD*) per il rilascio dell'app ogni volta che viene eseguito un commit o una pull request sul branch principale della repository Github, offrendo anche *deployment* di anteprima non disponibili al pubblico.

Tabella dei contenuti

Parametri di simulazione interattivi

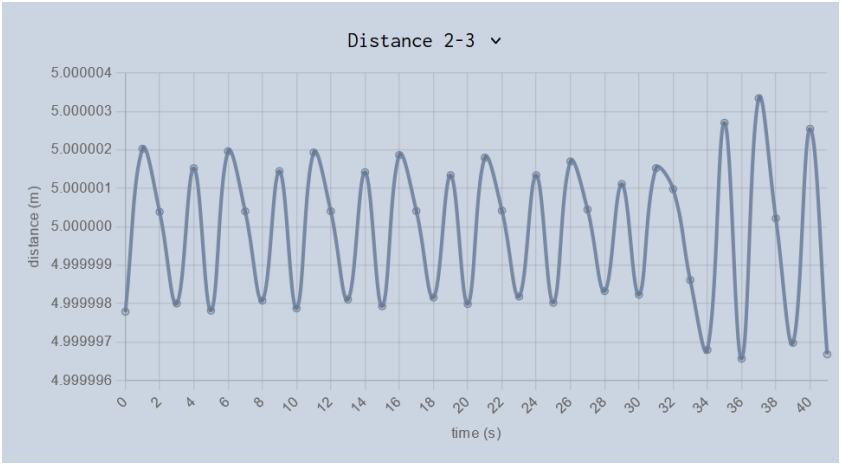
- Regolazione dinamica del **numero di veicoli** e dell'**obbiettivo di distanza** tra loro.
- Impostazioni per tutti i parametri del modello: *Tau*, *Kp*, *Kdd* e *TimeHeadway*

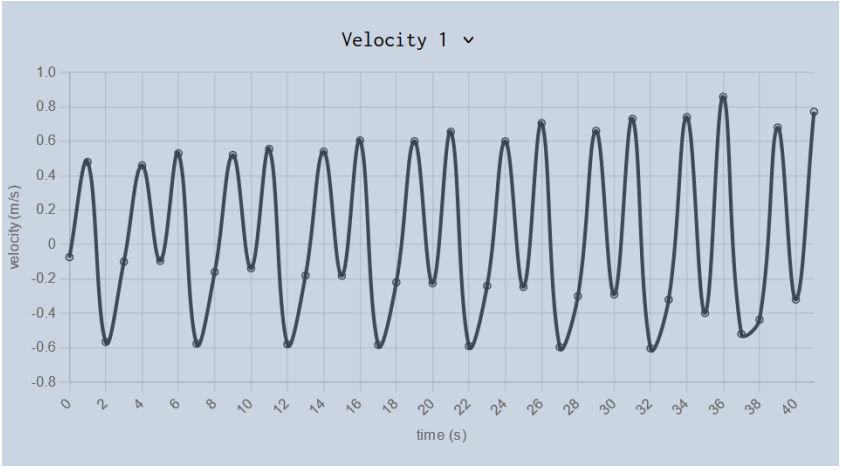
Parametri di simulazione interattivi



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

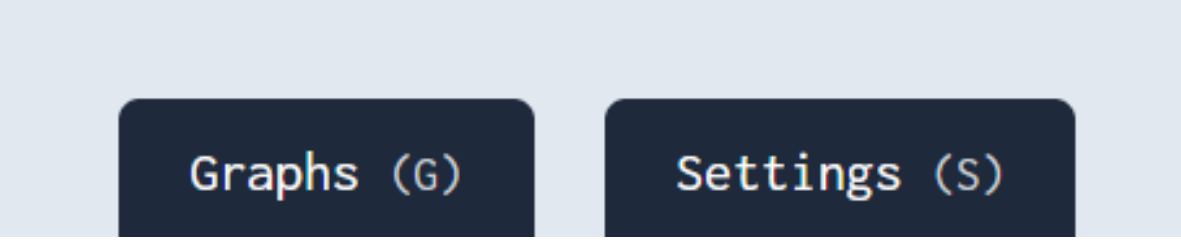
Visualizzazione dei dati campionati





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

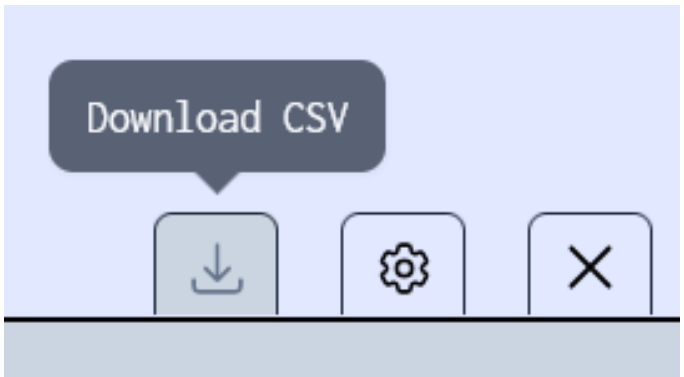
Shortcut da tastiera



- L'interfaccia è navigabile anche usando degli shortcut da tastiera:

`G`: attiva/disattiva la tab dei grafici
`S`: attiva/disattiva la tab delle impostazioni
`SPAZIO`: riproduce/mette in pausa 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)
```

Dettagli implementativi

Tutto il codice e la storia dello sviluppo sono disponibili pubblicamente sulla nostra *repository GitHub* con licenza **MIT**.

Tabella dei contenuti

Demo

platooning-visualization.vercel.app