Proyecto Final TC SW

Javier Ribal del Río

2026-03-27

Tabla de contenidos

1 Contextualización de la tarea

El Hyperloop es una tecnología que se basa en la eficiencia energética; para ello se introduce lo que a día de hoy conocemos como *booster*, un módulo de infraestructura que confiere energía al pod para su propulsión. Al externalizar la potencia de la cápsula al carril, se logra un sistema más verde y sostenible, reduciendo el peso del vehículo y optimizando el consumo eléctrico mediante una gestión inteligente de la energía.

A lo largo de los años en *Hyperloop-UPV* hemos tratado de desarrollar nuestro propio *booster*, uno de los problemas que identificamos a la hora de desarrollarlo es que para poder hacer pruebas era necesario disponer del POD completo. Por ello, este año hemos diseñado una bancada de pruebas[1] que nos permite validar el funcionamiento del *booster* de forma sencilla.

1.1 Fundamentos físicos

1.1.1 Componentes

Podemos dividir la bancada de pruebas en 4 componentes:

- **Carro *booster*** (o Carro): módulo de tracción móvil compuesto por un bastidor de cuatro ruedas y un EMS[2] que es propulsado por la sección
- ***Track***: vía de 50 metros por la que circula el Carro *booster*
  - **Sección *booster***: sección del *track* por la que el al pasar Carro *booster* lo impulsa.
- **Armario *booster***: lugar donde se almacenan los supercondensadores y las resistencias que generan el campo magnético que impulsa al Carro *booster* en la Sección *booster*

1.1.2 Variables

Trabajaremos como todas la variables asumiendo que son escalares.

1.1.2.1 Carro *Booster*

- Posición: ubicación física del Carro en el *Track*. s medida en m y $s \in [0, 50]$.
  - Asumiremos que la Sección *booster* se encuentra ubicada entre $s = 2$ y $s = 4$
- Velocidad: cambio de la posición del Carro respecto al tiempo. v medida en km/h
- Aceleración: cambio de la velocidad del Carro respecto al tiempo. a medida en m/s^2
- Masa: masa total del Carro *booster*. m medida en kg
- Fuerza: fuerza neta aplicada sobre el Carro, resultado del empuje del *booster* y de las fuerzas de frenado u oposición al movimiento. F medida en N. Por convenio, $F > 0$ si actúa en el sentido de avance (dirección creciente de s) y $F < 0$ en caso contrario.

1.1.2.2 Armario *Booster*

- Tensión: diferencia de potencial eléctrico suministrada por el sistema de almacenamiento de energía. V medida en V
- Intensidad: corriente eléctrica que circula por el sistema para generar el campo magnético del *booster*. I medida en A

1.1.2.3 Otras consideraciones

Asumiremos que:

[1] Sistema estructural diseñado para el montaje rígido de prototipos o componentes, equipado con sensores para la medición de parámetros operativos y la simulación de condiciones de carga controladas.

[2] Un imán.

- El movimiento únicamente es horizontal

$$
\begin{aligned}
g &= 9.81\,\mathrm{m/s^2} \\
F &= m \cdot a \\
|\vec{N}| &= |\vec{P}| = mg \\
F_{\text{rozamiento}} &= \mu|\vec{N}| \\
\mu &= \begin{cases} 0, & \text{si el Carro no está frenando} \\ 0.5, & \text{si el Carro está frenando} \end{cases}
\end{aligned}
$$

1.1.3 Funcionamiento

- Partiremos de un estado inicial de espera `IDLE` donde $s = 0; a = 0; v = 0; V = 0; I = 0;$
- Se dará la orden de precarga `PRECHARGE` donde todas las variables se mantendrán a 0 menos V que empezará a aumentar hasta alcanzar $V = 400$
- Cuando $V = 400$ entramos al estado de `READY`, donde la bancada está lista para operar
- Cuando se inicie la prueba, el carro adquirirá una $v_{\text{base}} = 4$ que mantendrá en todo momento hasta que frene. `RUNNING` $a = 0; V = 400$
- Al entrar en la Sección *Booster* la velocidad y aceleración del Carro se verán incrementadas. `BOOSTING`. Durante el funcionamiento del *Booster* la Intensidad se modificará
- Una vez haya salido de la Sección *Booster* ($s > 4$) mantendrá una velocidad constante de $v = 25$ hasta que el Carro llegue al final del *track* ($s = 50$) o bien sea frenado. `RUNNING`
- Frenado: al lanzar la orden de Frenado `BRAKE` el Carro comenzará a disminuir su velocidad hasta $v = 0$ o llegar al final del *track*.
  - Si el Carro llega a detenerse antes de alcanzar el final del *track* ($s < 50$), se informará mediante un mensaje
  - En caso de que el Carro llegue al final del *track* con $v > 0$, será detenido por el *mechanical stopper*[3], también se informará con un mensaje.
- Una vez el Carro reste detenido `STOPPED` todas las variables de mantendran a 0 salvo la posición.

[3] Plancha de metal que asegura que el Carro *booster* no salga del *track*.

2 Objetivos de aprendizaje

El objetivo principal de la tarea es poner en práctica el contenido aprendido durante el *training center* de Software, el diseño de UI dinámicas para el control de telemetría del vehículo. Asimismo, el proyecto se realizará en colaboración con otros subsistemas, para potenciar el trabajo en equipo y la comunicación entre los subsistemas, simulando así un *miniequipo* de Hyperloop.

3 Descripción general de la tarea

El objetivo del proyecto final consiste en desarrollar una bancada de pruebas para el *booster* (de ahora en adelante bancada *booster*). Los alumnos de los diferentes subsistemas del Training Center tienen asignadas diferentes tareas interrelacionadas entre sí, donde será necesario que todos cooperen de manera coordinada para cumplir con los requisitos técnicos y alcanzar el objetivo final de la tarea.

3.1 Descripción de la tarea de Software

Se deberá implementar un emulador de la bancada *booster*, el emulador estará compuesto por dos partes:

- Frontend: se conecta al backend y muestra datos sobre la bancada
- Backend: emulación de la bancada *booster*. (**Se permite hacer uso del backend de ejemplo**)

3.2 Evaluación de la tarea

Los requesitos mínimos para evaluar la tarea es realizar un frontend incluyendo las 5 funcionalidades detalladas en el apartado anterior.

Aquellos que deseen aspirar a nota extra podrán entregar también su propio backend que cumpla con las especificaciones de la tarea.

4 Frontend

La *Graphical User Interface* (GUI) o frontend, debe de conectarse al backend utilizando los protocolos `http` y `websockets`.

4.1 Especificacciones de la GUI

1. Una sección con gráficas que nos permita visualizar las siguientes varibles

- V
- a
- v
- F (Habrá que calcularla en el Frontend. Usar 2ª Ley de Newton)
- I
- Además, de una cronograma con los estados establecidos.

`IDLE`, `PRECHARGE`, `READY`, `RUNNING`, `BOOSTING`, `BRAKING` y `STOPPED`

2. Una sección que nos permita mediante botones enviar ordenes al simulador

- `PRECHARGE`: inicia la precarga
- `START`: inicia el movimiento del Carro, se debe de especificar la masa
- `BRAKE`: frena el Carro
- `RESET`: reinicia el simulador

3. Una sección que nos permita ver los mensajes que envía el simulador al *Front*, los mensajes harán referencia al estado del simulador, por ejemplo : *V = 400V precharge completed sucessfully*

4. Una sección que nos permita calcular la distancia óptima para presionar el freno mediante una llamada a API

5. Inclusión de un modelo 3D del Carro *Booster*

4.2 Implementación

4.2.1 Notas Generales

- Se debe desarollar una aplicación Web utilizando React
- Se debe utilizar preferiblemente Typescript o en su defecto JavaScript
- Se recomienda el uso de librerías de estilo como *tailwind* o *bootsrap*, así mismo se recomienda el uso de librerías de componentes, en especial *shadcn*
- El frontend deberá validar los inputs antes de enviarlos

4.2.2 Gráficas (1) y Mensajes (3)

El Backend expondrá en el puerto `5001` un protocolo `websockets` con ruta `backend/stream` que se corresponderá con el flujo de datos envidados del cliente a servidor; es decir utilizaremos `websockets` únicamente para leer datos. De la información transmitida por `ws` obtendremos tanto los datos que habrá que mostrar en la gráfica, como los mensajes que se debén de mostrar en el área de mensajes.

WebSocket endpoint: `ws://localhost:5001/backend/stream`

La comunicación por websockets estará, codificada como JSON, del cual distinguiremos 2 formatos, uno para recepción de los datos y otro para los mensajes del servidor.

Formato datos:

```
{
  "topic": "data",
  "payload": {
    "timestamp": "2026-03-27T10:15:32.125Z",
    "state": "RUNNING",
    "position_m": 3.42,
    "velocity_kmh": 18.7,
    "acceleration_ms2": 2.15,
    "mass_kg": 40,
    "voltage_v": 400,
    "current_a": 125.4
  }
}
```

Formato mensajes:

```
{
  "topic": "message",
  "payload": {
    "type": "info",
    "content": "Precharge started"
  }
}
```

Los tipos de mensajes pueden ser:`info`, `critical`, `error` y `success`

En el caso de formato de datos los paquetes llegarán con una frecuencia de 4 Hz, debido a la gran afluencia de paquetes se recomienda mantener un histórico limitado (por ejemplo, los últimos 5–10 segundos de datos) para la representación gráfica, eliminando los valores más antiguos para evitar problemas de rendimiento.

4.2.3 Botones (2)

Al ser presionados mandarán una orden utilizando una petición `HTTP-POST` al backend utilizando el formato especificado. El botón de `START` debe de estar acompañado de un `input` numérico que permita especificar la massa del carro.

La orden se dirigirá al siguiente enpoint `/api/command` la cual estará en el puerto `8001`. El servidor tras recibir la orden devolvera el código 200 indicando que todo ha sido correcto y se enviará un mensaje via `websockets`.

El servidor responderá con:

- `200 OK` si el comando se ha aceptado correctamente
- `400 Bad Request` si el comando es inválido

El resultado de la acción se notificará mediante un mensaje vía `websockets`.

Formato petición HTTP:

```
POST /api/command
Content-Type: application/json

{
  "command": "PRECHARGE"
}
```

(en el caso de la orden `START`)

```
POST /api/command
Content-Type: application/json
```

```
{
  "command": "START",
  "payload": {
    "mass": 40
  }
}
```

En caso de error (`400 Bad Request`), el frontend deberá mostrar un mensaje informativo al usuario indicando que la operación no se ha podido realizar.

4.2.4 *Cálculo* de la posición óptima para presionar el freno (4)

Sección con un formulario que permitirá indicar el peso del vehículo y a cuánta distancia se desea acabar del fin de la vía. Al presionar el botón de *submit* el frontend realizará una petición tipo `GET - HTTP` a `/api/calculate` con los *query parameters* `m` y `d`, que se corresponden con la masa del Carro y la distancia deseada, respectivamente.

El servidor devolverá una respuesta en formato JSON con la posición óptima en la que se debe iniciar el frenado. El frontend deberá procesar dicha respuesta y mostrar el resultado de forma clara al usuario.

Formato de respuesta esperado:

```
{
  "braking_position_m": 37.5
}
```

4.2.5 Inclusión del modelo 3D (5)

Se debe incluir un modelo 3D del Carro *booster* diseñado por el subsistema de *Mechanics*, debe de ser interáctivo, no debe de tratarse de una fotografía. [4]

> **Nota sobre la velocidad del backend de ejemplo**
>
> El backend de ejemplo no corre en tiempo real: la simulación transcurre **5× más lenta** que el tiempo real (`SIM_SPEED = 0.2`). La frecuencia de emisión por WebSocket se mantiene en 4 Hz, pero los valores de posición, velocidad, etc. avanzan más despacio de lo que cabría esperar físicamente. Esto es intencionado para facilitar las pruebas interactivas — el frontend no necesita hacer ningún ajuste al respecto.

4.3 Recomendaciones

- El objetivo de la tarea es diseñar un panel de control, por lo que sería interesante que haya una única vista, donde no se pueda hacer *scroll* que lo concentre todo
- Se recomienda hacer uso de los estilos como herramienta para resaltar los eventos que suceden en el simulador, por ejemplo si el Carro, se estrella con el *mechanical stopper*, podría ponerse el fondo en rojo para indicar que ha colisionado.

[4] No se incluye una descripción más extensa porque el desafio de este apartado es ver, vuestras habilidades para seleccionar la forma óptima.

5 Backend (VOLUNTARIA REALIZACIÓN)

El backend será el encargado de emular el comportamiento de la bancada *booster*, gestionando tanto la lógica de estados como la generación de datos de telemetría y la comunicación con el frontend.

El backend deberá implementarse preferiblemente en **Rust** o **Go**, o en su defecto en **Java**. Queda a criterio del alumno la elección del lenguaje dentro de estas opciones.

El backend deberá exponer dos tipos de comunicación:

- `WebSockets`: envío continuo de datos y mensajes al frontend
- `HTTP`: recepción de órdenes y cálculo de la posición óptima de frenado

5.1 Especificaciones del Backend

5.1.1 Arquitectura general

El backend deberá estar compuesto por:

- Un servidor `WebSocket` en el puerto `5001` para el envío de datos en tiempo real
- Un servidor `HTTP` en el puerto `8001` para la recepción de comandos y cálculo

El sistema deberá funcionar como una **máquina de estados**, donde las transiciones estarán controladas por las órdenes recibidas.

5.2 Máquina de estados

El backend deberá implementar los siguientes estados:

`IDLE`, `PRECHARGE`, `READY`, `RUNNING`, `BOOSTING`, `BRAKING` y `STOPPED`

5.2.1 Transiciones

- `IDLE` → `PRECHARGE` mediante comando `PRECHARGE`
- `PRECHARGE` → `READY` cuando $V = 400$
- `READY` → `RUNNING` mediante comando `START`
- `RUNNING` → `BOOSTING` cuando $2 \leq s \leq 4$
- `BOOSTING` → `RUNNING` cuando $s > 4$
- `RUNNING` → `BRAKING` mediante comando `BRAKE`
- `BOOSTING` → `BRAKING` mediante comando `BRAKE`
- `BRAKING` → `STOPPED` cuando $v = 0$
- `Cualquier estado` → `IDLE` mediante comando `RESET`: todas las variables vuelven a su valor inicial ($s = 0$, $v = 0$, $a = 0$, $V = 0$, $I = 0$, $m = 0$)

Los comandos únicamente podrán ser enviados por el frontend

No se permitirán transiciones no definidas.

5.3 Generación de datos

El backend deberá generar y enviar datos de forma continua mediante `WebSockets`.

- Frecuencia: **4 Hz**
- Cada mensaje representará el estado instantáneo del sistema

5.3.1 Variables a generar

- `position_m`
- `velocity_kmh`
- `acceleration_ms2`
- `mass_kg`
- `voltage_v`
- `current_a`
- `state`
- `timestamp`

El campo `timestamp` deberá estar en formato **ISO 8601 (UTC)**.

5.4 Lógica física simplificada

Se deberá respetar el siguiente comportamiento:

- En `PRECHARGE`: incremento progresivo de V hasta 400 (Incremento de 25 V por tick, es decir, cada 250 ms; la precarga completa dura 16 ticks = 4 segundos)
- En `RUNNING`: velocidad constante $v = 4\,\text{km/h}$ (antes del booster)
- En `BOOSTING`: la aceleración se recalcula en cada tick para garantizar que el Carro alcance exactamente $v_f = 25\,\text{km/h}$ al llegar a $s = 4\,\text{m}$:

  $$a = \frac{v_f^2 - v^2}{2\,(4 - s)}$$

  donde v y s son la velocidad (en m/s) y posición actuales. La fuerza es $F = m \cdot a$. **Cuando $s \geq 4$, no calcular a — fijar directamente $v = 25\,\text{km/h}$ y transicionar a `RUNNING`** para evitar división por cero. La intensidad sigue el perfil:

  $$I(s) = I_{\max} \cdot \sin\left(\frac{\pi\,(s-2)}{2}\right), \quad I_{\max} = 200\,\text{A}$$

- En `RUNNING` (post-booster): velocidad constante $v = 25\,\text{km/h}$, $a = 0$, $I = 0$
- En `BRAKING`: $F_{\text{brake}} = 196\,\text{N}$, $a = -F_{\text{brake}}/m$. En cada tick ($\Delta t = 0.25\,\text{s}$) se actualiza:

  $$v \leftarrow v + a\,\Delta t \qquad s \leftarrow s + v\,\Delta t$$

  Si $v \leq 0 \rightarrow$ transición a `STOPPED`. Si $s \geq 50 \rightarrow$ *mechanical stopper*.
- En `STOPPED`: todas las variables a 0 salvo posición

No es necesario implementar un modelo físico complejo, pero sí coherente con las reglas anteriores.

Velocidad de simulación

El backend no corre en tiempo real. El paso de tiempo físico se escala con un factor fijo `SIM_SPEED = 0.2`, de modo que la simulación transcurre **5× más lenta** que el tiempo real. La frecuencia de emisión por WebSocket se mantiene en 4 Hz.

$$\Delta t_{\text{físico}} = \Delta t \times 0{,}2 = 0{,}05\,\text{s/tick}$$

Esto proporciona aproximadamente 33 segundos de margen en el tramo post-booster para enviar la orden de frenado, facilitando las pruebas interactivas.

5.5 Comunicación WebSocket

El servidor deberá emitir mensajes en formato JSON siguiendo los dos tipos definidos:

5.5.1 Datos

```
{
  "topic": "data",
  "payload": {
    "timestamp": "2026-03-27T10:15:32.125Z",
    "state": "RUNNING",
    "position_m": 3.42,
    "velocity_kmh": 18.7,
    "acceleration_ms2": 2.15,
    "mass_kg": 40,
    "voltage_v": 400,
    "current_a": 125.4
  }
}
```

5.5.2 Mensajes

```
{
  "topic": "message",
  "payload": {
    "type": "info",
    "content": "Precharge started"
  }
}
```

Los tipos posibles son `info`, `success`, `error` y `critical`. El backend deberá emitir un mensaje en cada transición de estado:

Evento	`type`	`content` (ejemplo)
Comando `PRECHARGE` recibido	`info`	`"Precharge started"`
$V = 400$ alcanzado	`success`	`"V = 400V precharge completed successfully"`
Comando `START` recibido	`info`	`"Booster test started. Mass: 40 kg"`
Carro entra en Sección *Booster*	`info`	`"Booster section entered"`
Carro sale de Sección *Booster*	`success`	`"Boost completed. Velocity: 25 km/h"`
Comando `BRAKE` recibido	`info`	`"Braking initiated"`
Carro detenido ($v = 0$, $s < 50$)	`success`	`"Cart stopped at s = 12.3 m"`
Carro llega al *mechanical stopper*	`critical`	`"Cart reached mechanical stopper at s = 50 m"`
Comando no válido para el estado actual	`error`	`"Command BRAKE not allowed in state IDLE"`
Comando `RESET` recibido	`info`	`"System reset"`

5.6 Comunicación HTTP

El servidor HTTP escuchará en el puerto `8001` y expondrá dos endpoints.

5.6.1 `POST /api/command`

Recibe un comando del frontend. Si el comando no es válido para el estado actual, responde `400 Bad Request`. En caso contrario responde `200 OK` y emite el mensaje correspondiente por WebSocket.

Comandos aceptados:

Comando	Estado requerido	Payload adicional
`PRECHARGE`	`IDLE`	—
`START`	`READY`	`{ "mass": <número> }`
`BRAKE`	`RUNNING` o `BOOSTING`	—
`RESET`	Cualquiera	—

5.6.2 `GET /api/calculate`

Calcula la posición óptima para iniciar el frenado. Recibe dos *query parameters*:

- `m`: masa del Carro en kg
- `d`: distancia deseada al final del *track* en m

$$d_{\text{brake}} = \frac{v_0^2 \cdot m}{2\,F_{\text{brake}}} \qquad s_{\text{brake}} = (50 - d) - d_{\text{brake}}$$

con $v_0 = 25\,\text{km/h}$ (convertido a m/s) y $F_{\text{brake}} = 196\,\text{N}$.

Respuesta:

```
{
  "braking_position_m": 37.5
}
```

Si los parámetros son inválidos (negativos, ausentes, o $s_{\text{brake}} < 0$), responder `400 Bad Request`.

6 Uso del backend de ejemplo

Para los alumnos que realicen únicamente el frontend, se proporciona un backend de ejemplo ya compilado y listo para ejecutar, sin necesidad de instalar Python ni ninguna dependencia.

6.1 Descarga

Sistema operativo	Enlace
Windows	Descargar backend.exe
Linux	Descargar backend

6.2 Ejecución

6.2.1 Windows

Haz doble clic sobre `backend.exe` o ejecútalo desde una terminal:

```
backend.exe
```

6.2.2 Linux

Otorga permisos de ejecución y lánzalo:

```
chmod +x backend
./backend
```

6.3 Puertos

Una vez en marcha, el backend expone dos servicios de forma simultánea:

Servicio	URL
WebSocket (telemetría)	`ws://localhost:5001/backend/stream`
HTTP (comandos y cálculo)	`http://localhost:8001`

El frontend debe estar lanzado **después** de que el backend esté corriendo. Asegúrate de que los puertos `5001` y `8001` no están ocupados por otro proceso.

7 Entrega

La entrega del proyecto se realizará a través de un **repositorio de GitHub** que deberá cumplir los siguientes requisitos:

- El repositorio debe contener **únicamente el proyecto** — sin archivos innecesarios, sin el binario del backend de ejemplo descargado, y sin carpetas de dependencias (`node_modules`, entornos virtuales, etc.). Se recomienda incluir un `.gitignore` adecuado.
- El repositorio debe incluir un `README.md` con instrucciones claras para instalar y ejecutar el proyecto.
- Se enviará el enlace al repositorio al responsable del subsistema de Software.

7.1 Plazos

Hito	Fecha
Plazo para añadir cambios al repositorio	**12 de mayo de 2026**
Exposición del proyecto	**14 de mayo de 2026**

A partir del **12 de mayo** no se tendrán en cuenta los cambios realizados en el repositorio para la evaluación. La exposición consistirá en una demostración en vivo del proyecto y una breve explicación de las decisiones de implementación tomadas.