Avance Semanal: Desafío Técnico Eolian Auto Solar

Integrante: Sebastián Morales (*sebmorales@ug.uchile.cl*)

17 de julio de 2025

Semana # 1	**Fecha: 23 de junio de 2025 - 29 de junio de 2025**
Actividades realizadas:	Revisé la documentación del desafío y me imaginé una planificación inicial.
Descubrimientos y aprendizajes clave:	No hubo porque no pude avanzar esa semana :(.
Bloqueos o dificultades encontradas:	No hubo porque no pude avanzar esa semana :(.
Plan para la próxima semana:	Investigar más sobre que es una red CAN y cuál es el formato de los mensajes que se envían por ella. Investigar como hacer diagramas UML para comenzar con la arquitectura.

Semana # 2	**Fecha: 30 de junio de 2025 - 6 de julio de 2025**
Actividades realizadas:	Aquí hubo muchos avances, principalmente en entendimiento, diseño y un intento de producto mínimo viable. Generé un diagrama UML de los diversos componentes del sistema y un diagrama de secuencia para ilustrar el flujo de un mensaje en el sistema. Hice un programa main.cpp que logró generar una interfaz para ingresar frames CAN y mostralos en la misma terminal
Descubrimientos y aprendizajes clave:	Entendí, viendo videos en youtube en su mayoría, como funciona el sistema CAN (también entendí que no es exclusivo de automóviles, sino que se usa en muchos sistemas embebidos). Comprendí su formato y en base a eso es como los guardé en el programa que hice. También entendí como funcionan los diagramas UML y como hacerlos con una herramienta llamada PlantUML.
Bloqueos o dificultades encontradas:	Tuve problemas para mostrar los frames en la terminal, porque la UI los imprimía en la misma terminal donde estaba escribiendo,

	por lo que generaba flickering y no se podía ver bien (aunque el recibir y parsear los frames estaba funcionando).
Plan para la próxima semana:	En base a lo que aprendí, pienso que es mejor tener una interfaz gráfica para mostrar los frames, espero mostrarlos en una página web con html, css y javascript. Con esto, creo que es mejor desechar la idea de clases en C++ para cada sistema y es mejor tener un solo código en C++ que genere los frames y otro en JS que los muestre en la web.

Semana # 3 (y unos días adicionales)	**Fecha: 7 de julio de 2025 - 17 de julio de 2025**
Actividades realizadas:	En estos días es donde más avances generé. Creé una interfaz web básica con HTML y JS que muestra los frames recibidos con un timestamp. Para pasar los frames entre el programa en C++ y la interfaz web, utilicé un archivo /tmp/frames.fifo como pipe. Cambié todo el código en C++, ahora tiene un solo código sin clases, el cual lee los frames desde stdin o los genera aleatoriamente según el modo que se elija. Programé que también al enviar los frames se guardaran en un archivo can_log.txt. Cuando se actualiza un frame se ve que bytes fueron los modificados. Se puede cambiar la cantidad máxima de frames que se muestran en la interfaz web para no sobrecargar la página.
Descubrimientos y aprendizajes clave:	Aprendí más de JS y el uso de npm y hacer Makefiles más eficientes.
Bloqueos o dificultades encontradas:	Me costó encontrar los tipos específicos de algunas variables, en especial el de la estructura de datos en JS para mostrar los frames CAN en la interfaz web. También me costó entender como hacer que la interfaz web se actualizara automáticamente cuando se recibían nuevos frames.
Plan para la próxima semana:	Como futuras mejoras, añadiría una opción para controlar la cantidad de frames que se muestran en la interfaz web. Añadiría también, en base a la ID, un mapeo para mostrar en pantalla el nombre del dispositivo que envió el frame.