

# Reporte Final Concentración en Movilidad Inteligente (MR3004C)

Mariana Manjarrez Lima  
*Tecnológico de Monterrey,*  
*School of Engineering and Sciences*  
Puebla, México  
A01735160@tec.mx

Abraham Moro Hernández  
*Tecnológico de Monterrey,*  
*School of Engineering and Sciences*  
Puebla, México  
A01737333@tec.mx  
orcid.org/0009-0001-8001-3252

Pedro García Millán  
*Tecnológico de Monterrey,*  
*School of Engineering and Sciences*  
Puebla, México  
A01736798@tec.mx

Iván Valdez del Toro  
*Tecnológico de Monterrey,*  
*School of Engineering and Sciences*  
Puebla, México  
A01424018@tec.mx

Franco Abraham Díez  
*Tecnológico de Monterrey,*  
*School of Engineering and Sciences*  
Puebla, México  
A01737394@tec.mx

Yonathan Romero Amador  
*Tecnológico de Monterrey,*  
*School of Engineering and Sciences*  
Puebla, México  
A01737244@tec.mx

**Resumen**—Este reporte documenta el desarrollo de un sistema de navegación autónoma implementado en plataformas experimentales para la emulación de un automóvil realista. Se aplicaron metodologías de control y estimación de estados (Pure Pursuit y fusión de sensores), validando los algoritmos en un vehículo a escala (Quanser QCar1). Paralelamente, se realizó el restablecimiento de un prototipo vehicular eléctrico (AMR1), implementando una arquitectura de red CAN bus descentralizada, dejándolo operativo para futuras aplicaciones de conducción autónoma a escala completa.

**Index Terms**—Vehículo autónomo, Vehículo eléctrico, ROS 2, Autonomous Mobile Robot (AMR1), Quanser QCar.

## I. INTRODUCCIÓN

La movilidad inteligente requiere que los vehículos sean capaces de percibir su entorno, interpretar la información sensorial y actuar en consecuencia para garantizar una conducción segura, eficiente y autónoma. En este contexto, la fusión de sensores se ha convertido en una técnica clave para mejorar la precisión y robustez de la percepción [1], al combinar múltiples fuentes de datos que compensan sus fortalezas y limitaciones individuales.

Este proyecto responde a la necesidad de desarrollar soluciones que integren tecnologías de electrificación, automatización y control en plataformas experimentales. El vehículo eléctrico disponible funge como banco de pruebas para enfrentar los desafíos reales de adquisición y procesamiento de datos en tiempo real dentro de un ecosistema de movilidad inteligente.

Dada la complejidad del sistema, el proyecto se comparte en dos propuestas principales de implementación que convergen hacia la recreación de un vehículo autónomo:

- **Quanser QCar (Cerebro de operación):** Plataforma a escala 1:10 que cuenta con la instrumentación necesaria

(LiDAR, cámara RGBD, IMU, encoder) para la implementación y validación de algoritmos de percepción y control.

- **Plataforma AMR1:** Vehículo eléctrico que permite operar bajo condiciones dinámicas y mecánicas próximas a un automóvil realista [2].

### I-A. Objetivos

El objetivo principal es el diseño e implementación de una arquitectura de fusión de sensores que permita a un vehículo experimental navegar de forma autónoma en un entorno controlado. Esto se logra combinando información de múltiples fuentes (LiDAR, encoder, IMU) para estimar con precisión la pose del vehículo y ejecutar algoritmos de seguimiento de trayectoria y detección de obstáculos en tiempo real.

## II. SISTEMA DE CONTROL Y TOMA DE DECISIONES

En la última década, la navegación autónoma ha trascendido el ámbito teórico para convertirse en una realidad tangible. El pilar fundamental que sostiene la autonomía es la capacidad de localización y estimación del estado: la habilidad de determinar la posición y orientación precisas dentro de un marco de referencia global [3].

La operación exitosa de un Vehículo Terrestre Autónomo (AGV) se fundamenta en la ejecución cíclica de cuatro etapas críticas: Percepción, Decisión, Control y Actuación [4]. Dentro de este esquema, la percepción constituye la base piramidal [3], [5]; sin una interpretación precisa del estado del vehículo, cualquier decisión de control subsecuente será errónea. Este trabajo se centra en la subtarea de localización, abordando el desafío de estimar la pose en un plano bidimensional mediante fusión de sensores [6].

### II-A. Problemática de los Sensores Individuales

La navegación en entornos reales presenta desafíos significativos debido a la incertidumbre intrínseca de los sensores.

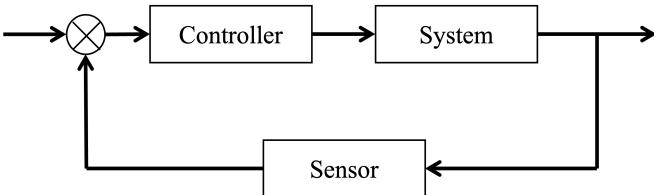


Figura 1: Diagrama de bloques del lazo de control cerrado implementado en el QCar.

El problema de estimación de estado se puede descomponer en tres limitaciones principales identificadas en la plataforma experimental:

- **Limitaciones de Estimación por estima (Dead Reckoning):** Sensores como los encoders e IMU permiten estimar la posición relativa mediante integración numérica. No obstante, este método sufre de *drift* (acumulación de error), donde pequeñas desviaciones en la medición se integran en el tiempo, provocando que la pose estimada diverja de la real.
- **Inexactitud y Ruido:** Sensores de referencia absoluta, como el GPS o sistemas de visión, suelen presentar ruido de medición, latencia o susceptibilidad a condiciones ambientales. En navegación autónoma, un error de posición de un metro es inaceptable y puede derivar en colisiones.
- **Restricciones de Hardware (QCar):** Al ser una plataforma integrada, el QCar presenta limitaciones para la modificación o adición de nuevos sensores, obligando a maximizar el rendimiento de los sensores preinstalados (IMU de 6 ejes, encoder, LiDAR, cámara RGBD, entre otros) mediante software.

## II-B. Estrategia de Solución

Implementar un algoritmo de fusión sensorial basado en el **Filtro de Kalman (KF)**. Este algoritmo sintetiza la información proveniente de la odometría y la IMU para generar una estimación recursiva y robusta del estado, capaz de reducir el error cuadrático medio y rechazar mediciones atípicas.

La arquitectura se ha desarrollado sobre el middleware ROS 2 usando la distribución Humble, lo que permite una gestión modular del flujo de datos en tiempo real [3], [7]. Finalmente, sobre la estimación de pose resultante, se ejecuta el algoritmo de control *Pure Pursuit*, permitiendo que el vehículo navegue autónomamente equilibrando la estabilidad lateral con la precisión del seguimiento de trayectoria.

## II-C. Plataforma Experimental: Quanser QCar

Dado que la validación directa en el vehículo a escala real conlleva riesgos operativos y costos elevados, se establece una etapa experimental intermedia utilizando el Quanser QCar. Esta plataforma a escala 1:10 no solo reduce la brecha entre simulación y realidad, sino que permite validar la arquitectura de software en un entorno seguro.

A diferencia de la implementación final en el AMR, el QCar actúa como un "gemelo escalado de comportamiento", permitiendo aislar y resolver los desafíos de estimación de

estado descritos anteriormente mediante una arquitectura de software modular basada en ROS 2.

### II-D. Objetivos de la Fase Experimental

El propósito fundamental de esta etapa es desarrollar y validar una arquitectura de navegación autónoma robusta en la plataforma a escala (QCar), que sea posteriormente escalable al vehículo 1:1 (AMR1). La estrategia se centra en la implementación de algoritmos de fusión sensorial y control de trayectoria que compensen las limitaciones de hardware intrínsecas de los sensores económicos.

**II-D1. Objetivos Específicos:** Para garantizar la transferibilidad de los resultados, se establecen los siguientes hitos técnicos:

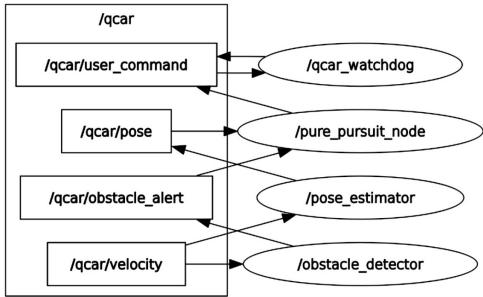
- **Arquitectura de Software en ROS 2:** Configurar el entorno de trabajo (middleware) y desarrollar los nodos de comunicación necesarios para la adquisición asíncrona de datos provenientes de los sensores del QCar (LiDAR, IMU y Encoder).
- **Fusión de Sensores (Estimación de Estado):** Implementar un algoritmo que fusione la odometría relativa (propensa al *drift*) con las mediciones iniciales, minimizando la covarianza del error para obtener una estimación precisa y estable de la pose global ( $x, y, \theta$ ).
- **Control de Seguimiento (Pure Pursuit):** Se integró la estimación de pose del vehículo con un controlador geométrico de seguimiento de trayectoria tipo *Pure Pursuit*. Este controlador selecciona un punto de referencia adelantado (*lookahead point*) sobre la ruta y calcula la curvatura de giro necesaria para alcanzarlo mediante relaciones geométricas entre la posición y orientación del vehículo y dicho punto. El lookahead se ajusta dinámicamente en función de la velocidad de referencia, permitiendo generar trayectorias suaves y estables mientras el QCar avanza sobre la ruta predefinida.
- **Validación Experimental:** Evaluar el desempeño del sistema mediante pruebas físicas en pista, comparando la trayectoria deseada contra la trayectoria real con el controlador (ground truth) y analizando la robustez del sistema ante el ruido de sensores.
- **Verificación de Escalabilidad:** Comprobar la modularidad del código desarrollado para asegurar su portabilidad a la plataforma AMR, validando que la lógica de estimación de pose y control funcione independientemente del hardware subyacente.

## III. IMPLEMENTACIÓN EXPERIMENTAL EN QCAR

### III-A. Entorno de Desarrollo y Ejecución

La implementación experimental se ejecutó principalmente en una laptop **ASUS ROG Strix G16 (Intel Core i9, 32 GB RAM, Ubuntu 22.04, ROS 2 Humble)**, donde se desarrollaron y corrieron todos los nodos de percepción, fusión sensorial y control utilizados para la navegación autónoma del QCar.

Adicionalmente, se empleó una laptop ASUS TUF Gaming (Ryzen, 16\,GB RAM,



**Figura 2:** Conexiones entre los nodos fundamentales durante el proceso de navegación autónoma

Windows~11 con WSL2 Ubuntu~22.04) como plataforma de apoyo para la ejecución secundaria de nodos de control, análisis de datos y depuración durante las pruebas de campo.

La capa de adquisición de datos del vehículo estuvo a cargo de un microcontrolador ESP32, encargado de la lectura del IMU de la marca Bosch BNO055 y de la comunicación serial con los nodos de navegación.

La implementación de la arquitectura de navegación autónoma se ejecuta sobre el middleware ROS 2, orquestando la interacción entre la percepción y el control. Toda la documentación relacionada a los códigos fuente desarrollados para esta implementación se encuentra disponible en el repositorio del proyecto (véase el Anexo A.).

### III-B. Desarrollo

El sistema se compone de cinco nodos fundamentales que operan de manera (vease esquema de conexión en la figura 2):

- **pose\_ekf\_qcar\_2:** Nodo de estimación de estado; fusiona datos iniciales y odometría para publicar la pose global.
- **pure\_pursuit\_node:** Controlador de trayectoria; calculando el ángulo de dirección necesario para seguir la referencia.
- **qcar\_lidar\_alert\_2:** Sistema de percepción reactiva; gestiona el paro de emergencia ante detección de una potencial colisión frontal.
- **qcar\_watchdog\_node:** Supervisor de estado; asegura el paro total del vehículo al finalizar la trayectoria o ante fallos del sistema.
- **trayectoria\_grabar\_csv\_node:** Utilería para la digitalización de rutas y generación de *ground truth*.

Bajo la arquitectura actual, primero se graba una trayectoria, (figura 4); mediante `pose_ekf_qcar_2.py` y `trayectoria_grabar_csv_node.py`, este nodo registra los puntos enviados desde la pose para elaborar un archivo csv que será la trayectoria de referencia del recorrido con el Qcar; el output por parte del grabador (figura 5), entrega un registro en X, Y, y Theta, aunque solamente se utilizan X, Y dentro del marco global para el seguimiento de

trayectoria.

Siguiendo la línea de preparación para la navegación, se realiza el encendido de la plataforma a escala 1:10 mediante:

- Encendido del Qcar; asegurarse que la batería tenga suficiente carga (entre 11 V y 12.4 V) para la realización óptima de pruebas.
- Conexión desde una computadora externa mediante ssh -X nvidia @ [IP] donde la IP es la correspondiente al Qcar utilizado para las pruebas.
- Entrar en modo super-usuario mediante sudo -s.
- Lanzamiento de nodos para publicación de los sensores del Qcar; el cual tienen dos variantes, dependiendo de la funcionalidad, ya que solo uno activa el control remoto (joystick).
- Para grabar la trayectoria se necesita el joystick, mediante el comando `ros2 launch qcar qcar_launch_modified.py`
- Para usar el controlador se necesita quitar la interferencia del control por lo que no se debe activar para tener un buen funcionamiento sin ruido, mediante el comando `ros2 launch qcar qcar_launch_sm_aim.py`

Teniendo la preparación anterior, debería poder observar a los sensores de la plataforma encendida (es más visual en el LiDAR). Al estar encendida correctamente, se puede comenzar el procedimiento de navegación.

Se inicia mediante la calibración de la pose mediante `pose_ekf_qcar_2` el cual marca el origen (0.0) de referencia global, esta coordenada inicial siempre es calibrada al punto en donde se encuentra el Qcar cuando se activa el comando `ros2 run sensores_kalman pose_ekf_qcar_2`.

Todos los vehículos de navegación autónoma realizan un seguimiento de un recorrido de referencia mediante alguna estrategia que permite al vehículo mantenerse próximo a la medida de referencia, mientras que el procesamiento del entorno lo hace mediante los sensores, que le permite evitar potenciales obstáculos para el automóvil.

Ocurre lo mismo ante el sistema que se documenta, se tiene una trayectoria de referencia grabada mediante `trayectoria_grabar_csv_node`, mientras que el seguimiento lo realiza el `pure_pursuit_node`.

El algoritmo utilizado permite que el vehículo pueda visualizar N-distancia de la referencia hacia delante, mientras que adapta la interacción del controlador mediante una ganancia K que hace el seguimiento adaptativo.

Continuando con la explicación de navegación autónoma; se deben tener cuatro nodos activos para hacer que el Qcar se mueva en el siguiente orden, tomando en cuenta que en las terminales el usuario debe asegurarse de tener el mismo ROS DOMAIN ID del Qcar:

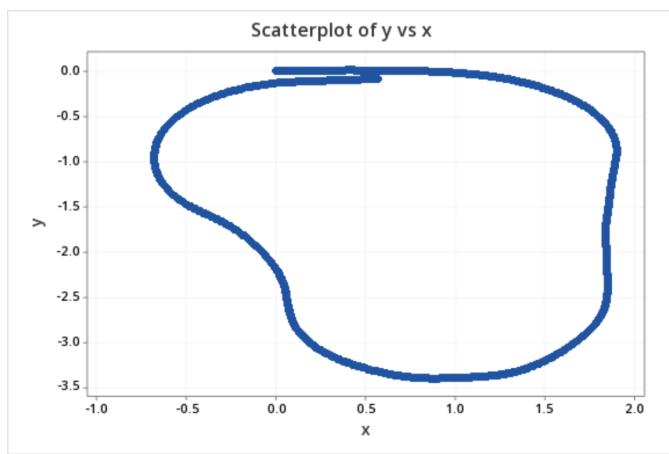
```
export ROS_DOMAIN_ID=[Número]
```

```

=====
POSE ESTIMATOR INICIADO
=====
Suscripto a:
- /qcar/velocity (Vector3Stamped)
- /imu/accel_raw (TwistStamped)
Publicando en:
- /qcar/pose (Vector3Stamped)
=====
Posición: X=0.000 Y=0.000 | Orientación: 0.0° | Vel: 0.000m/s

```

**Figura 3:** Visualización de nodos en ROS 2: Publicación de la pose estimada.



**Figura 4:** Diagrama de dispersión de la trayectoria de referencia registrada.

	x	y	yaw
83	0.01028	0.00016	0.01850
84	0.01460	0.00024	0.01955
85	0.01891	0.00033	0.01955

**Figura 5:** Estructura del archivo CSV generado por el nodo grabador.

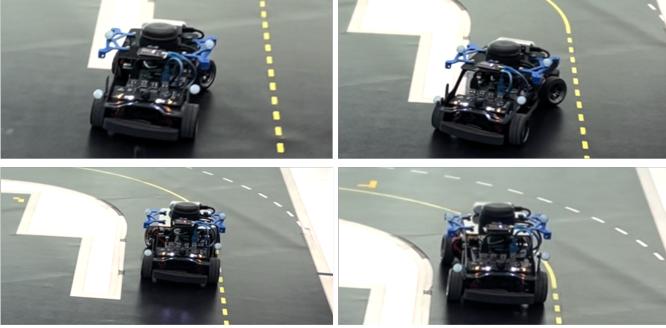


**Figura 6:** Inicio de la navegación autónoma en la plataforma QCar.

del entorno de ros del Qcar  
(revisar con "env | grep ROS") ]

- En terminal se activa el paro con detección de obstáculo del LiDAR con el comando `ros2 run sensores_kalman qcar_lidar_alert_2`, este nodo es de seguridad para evitar que el Qcar llegue a colisionar en caso de encontrar un obstáculo, ya que genera un cono de detección de 35 cm y con ángulo de 45° a menos de 2 m/s de operación, cuando aumenta la velocidad se abre a 60° el cono de observación. El LiDAR manda el mensaje booleano (true) por el tópico para evitar que siga el movimiento; una vez se retire el obstáculo, podrá seguir la trayectoria.
- En la segunda terminal se debe tener el nodo `ros2 run sensores_kalman qcar_watchdog_node`, que se encarga de supervisar los valores que se publican al nodo `user_command`, responsable de mandar los valores al Qcar, de tal manera que, cuando se pare el controlador se pueda detener el movimiento de la plataforma por completo.
- En la tercera terminal se actualiza la pose para iniciar la trayectoria en el origen, con el comando `ros2 run sensores_kalman pose_ekf_qcar_2`, y se recomienda siempre iniciar la trayectoria con la pose en X:0, Y:0, YAW:0.
- Por último, se activa el controlador, con el comando `ros2 run sensores_kalman pure_pursuit_node` (se utilizó la configuración del controlador `-p lookahead:=0.15 -p k_gain:=0.4 -p v_ref:=0.05`); este es el encargado de mover el Qcar. Debe considerarse que el código requiere que se le envíe un archivo .csv de la trayectoria de referencia, por lo que se agrega al comando `--ros-args -p path_csv:=/home/usuario/ruta_para_encontrar_el_csv/trayectoria_referencia.csv`. En caso de no proporcionar una trayectoria, se genera automáticamente un círculo con un radio modificable, `-p circle_radius:=0.0` en metros, y por la naturaleza del controlador, se pueden modificar tres valores para variar el comportamiento de la plataforma

y experimentar hasta encontrar el óptimo, ya sea directamente en el código o en la terminal, agregando después del path: `-p lookahead:=0.0 -p k_gain:=0.0 -p v_ref:=0.0`. Este nodo comienza a darle comandos de tipo avanza, frena o gira al vehículo, de modo que el seguimiento de trayectoria ha comenzado 6.



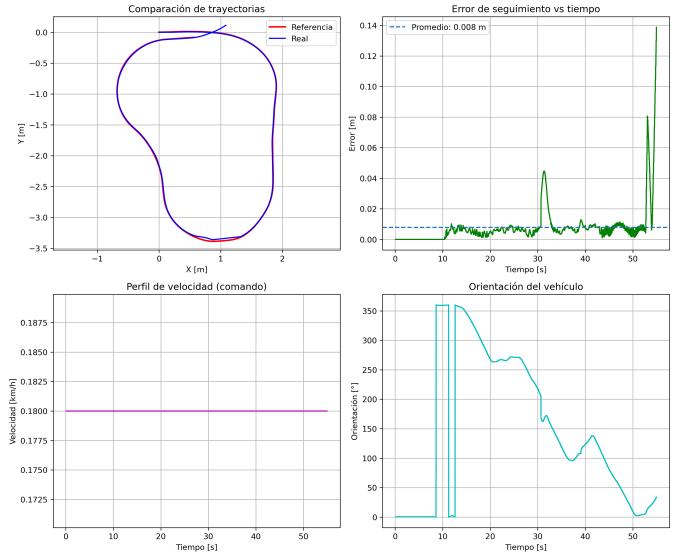
**Figura 7:** Comparativa visual: Trayectoria de referencia vs. Trayectoria real estimada.

El IMU y el encoder detectan movimiento por parte del vehículo, información que llega limpia por el filtro de Kalman integrado en el IMU BNO055 y se publica en los tópicos `/qcar/accel_raw` y `/qcar/velocity`. Esta es la información de entrada.

De tal manera que se transforman para el nodo `qcar/pose/`. Este último retroalimenta a `pure_pursuit_node` para que este pueda estimar que tan lejos está de la trayectoria de referencia, y pueda mandar las instrucciones concretas para que el vehículo haga las modificaciones necesarias para alcanzar el objetivo. De ese modo, se hace un ciclo de interacción entre `pure_pursuit`, y la pose. Sin embargo, la captación de información junto a los sensores tienen un error, el cuál se puede denominar como error acumulado (que es, de modo efectivo, lo que debe ser compensado por la ganancia tanto del filtro de Kalman, como el `pure pursuit`).

### III-C. Análisis de Resultados Experimentales

El desempeño del sistema de navegación autónoma se evalúa contrastando la odometría estimada en la prueba real con el controlador contra la trayectoria de referencia (Ground Truth). Se adjunta la variedad de resultados obtenida a partir de la misma configuración.

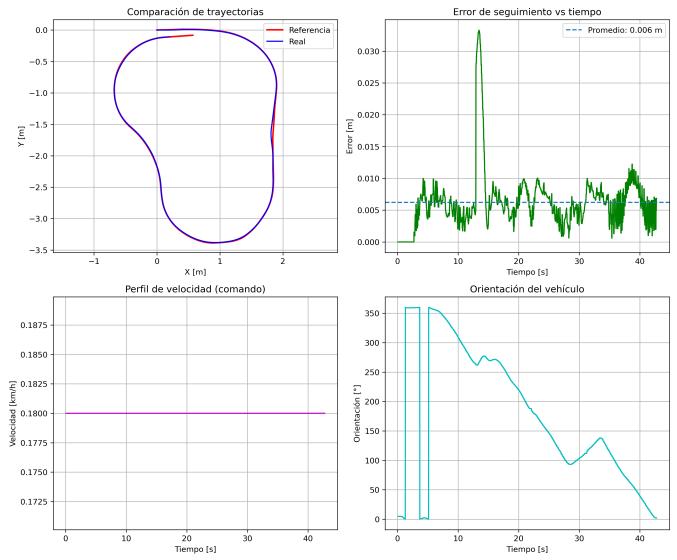


**Figura 8:** Resultados de recorrido, error medio, velocidad, y orientación, durante pruebas de autonomía

===== EVALUACIÓN DE TRAYECTORIA =====  
 Error promedio: 0.006 m  
 Error máximo: 0.033 m  
 Desviación estándar: 0.005 m  
 % dentro de 5cm: 100.0%  
 % dentro de 10cm: 100.0%  
 Trayectoria referencia: 918 puntos  
 Trayectoria real: 2134 puntos

Evaluación general: EXCELENTE

**Figura 9:** Análisis estadístico directo en la terminal sobre el recorrido en la figura 8.



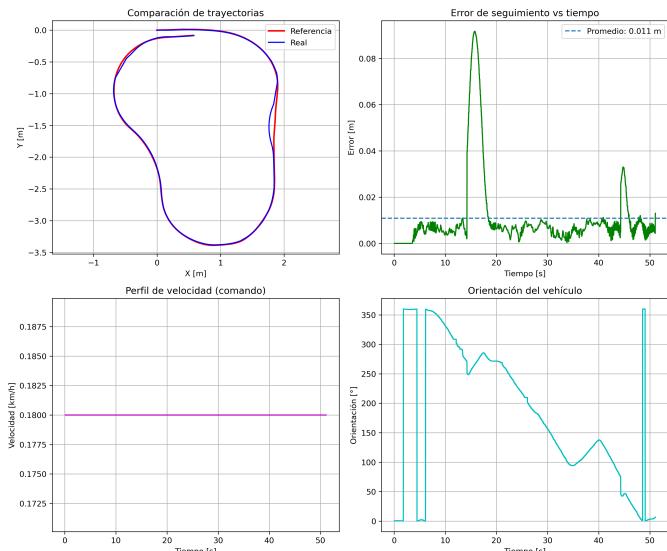
**Figura 10:** Resultados de recorrido, error medio, velocidad, y orientación, durante pruebas de autonomía

===== EVALUACIÓN DE TRAJECTORIA =====

Error promedio: 0.008 m  
Error máximo: 0.139 m  
Desviación estándar: 0.013 m  
% dentro de 5cm: 97.9%  
% dentro de 10cm: 99.6%  
Trayectoria referencia: 918 puntos  
Trayectoria real: 2743 puntos

Evaluación general: EXCELENTE

**Figura 11:** Análisis estadístico del recorrido en la figura 10.



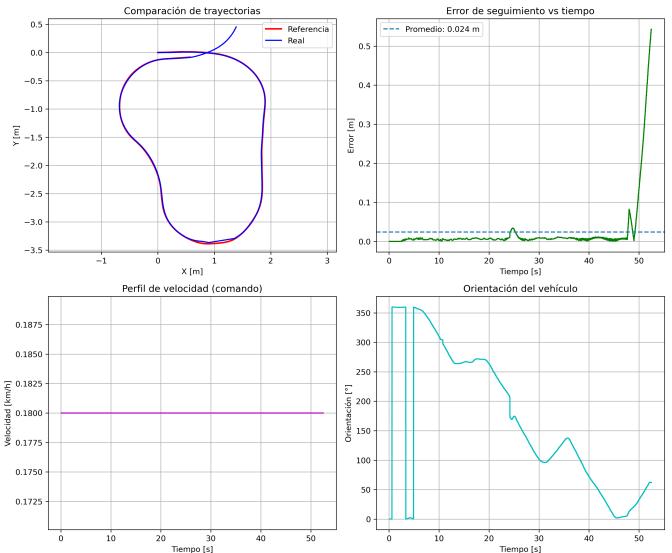
**Figura 12:** Resultados de recorrido, error medio, velocidad, y orientación, durante pruebas de autonomía

===== EVALUACIÓN DE TRAJECTORIA =====

Error promedio: 0.018 m  
Error máximo: 0.159 m  
Desviación estándar: 0.037 m  
% dentro de 5cm: 89.7%  
% dentro de 10cm: 92.6%  
Trayectoria referencia: 918 puntos  
Trayectoria real: 2243 puntos

Evaluación general: EXCELENTE

**Figura 13:** Análisis estadístico del recorrido en la figura 12.



**Figura 14:** Resultados de recorrido, error medio, velocidad, y orientación, durante pruebas de autonomía

===== EVALUACIÓN DE TRAJECTORIA =====

Error promedio: 0.011 m  
Error máximo: 0.092 m  
Desviación estándar: 0.017 m  
% dentro de 5cm: 94.6%  
% dentro de 10cm: 100.0%  
Trayectoria referencia: 918 puntos  
Trayectoria real: 2551 puntos

Evaluación general: EXCELENTE

**Figura 15:** Análisis estadístico del recorrido en la figura 14.

Al analizar las diferentes pruebas, se observa una convergencia satisfactoria entre la trayectoria ejecutada y la referencia. Sin embargo, se identifican dos comportamientos característicos del controlador *Pure Pursuit*:

- Suavizado de Curvas (Corner Cutting):** En los segmentos de mayor curvatura, se aprecia una ligera desviación intencional hacia el interior de la curva. Esto es consecuencia del parámetro *Lookahead Distance*; el algoritmo prioriza un punto objetivo adelantado para reducir el esfuerzo de control (cambios bruscos en la dirección), optimizando la suavidad del viraje a costa de una precisión milimétrica absoluta.
- Condición de Frontera (Salida):** Al final del recorrido, se presenta una discontinuidad en el seguimiento. Esto ocurre debido a la reducción de puntos disponibles en el búfer de anticipación (horizonte de predicción) conforme se acaba el archivo .csv. Esta singularidad no compromete la seguridad, ya que el nodo *qcar\_watchdog\_node* intercepta la finalización de la lista de puntos y ejecuta el protocolo de paro seguro antes de que se genere una inestabilidad.

Sin embargo, desde otro punto de vista, existe una propiedad repetible en todas las pruebas, tratándose de la configuración del controlador  $-p$  lookahead:=0.15  $-p$  k\_gain:=0.4  $-p$  v\_ref:=0.05. A pesar de que las entradas son relativamente similares, se puede denotar que los resultados no son idénticos, sino similares. Esto abre un debate con respecto a que factores fueron relevantes al momento del desarrollo de la trayectoria correspondiente.

Al comparar la figura 8 y 10 [8], donde el mayor esfuerzo de control se realizó en secciones distintas; se cree que puede haber diversas fuentes de variación para el seguimiento de trayectorias. Para resumir el origen de la problemática, se realiza la aplicación de un diagrama de Ishikawa (observese en la figura 16), donde se logra detectar cualitativamente que la variación esta originada por el medio donde se realizan (o realizarán) las pruebas, así como el vehículo mediante el cual se centra el proceso. En contraste con la experiencia empírica, es posible denotar los hallazgos del diagrama de Ishikawa, debido a que en muchas ocasiones, el avance de autonomía se vió delimitado por la plataforma en la que se realizaba el proyecto; o, un medio diferente (excluyendo obstáculos tridimensionales) podía inclusive parar completamente el procedimiento de conducción autónoma. Quizás el cambio de plataforma podría significar un mejor desempeño por parte del controlador, aunque también podría afectar en otras áreas de desempeño (como lo sugiere el diagrama de Ishikawa), mientras que el control del medio en realidad se trata del desafío principal, puesto que la práctica apunta a que el controlador debería poder funcionar a pesar de ciertas limitaciones ambientales. Claro, esto se sugiere de manera cualitativa.

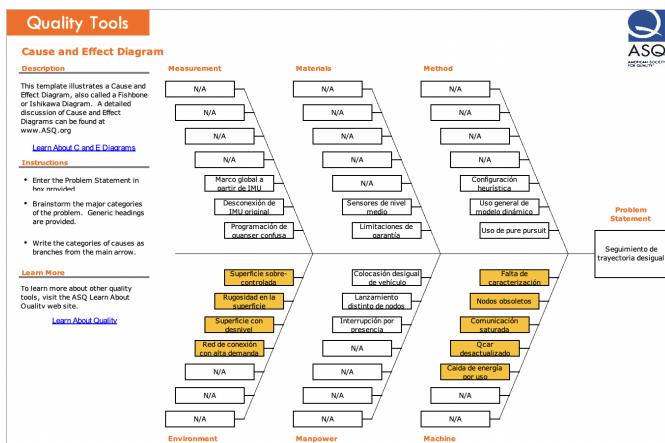


Figura 16: Diagrama de Ishikawa para rastreo de propiedades en el seguimiento de trayectoria

Cuantitativamente, se realiza un análisis a partir de los errores obtenidos de trayectoria, se estudiará la figura 10 por el esfuerzo de control, a fin de estudiar el controlador como un general, más allá del empirismo y el prospecto cualitativo. Para ello se adjuntan las siguiente pruebas estadísticas.

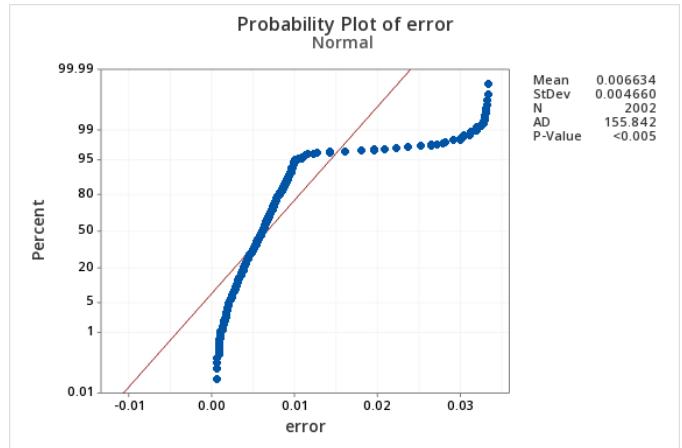


Figura 17: Prueba de normalidad para desempeño de seguimiento autónomo

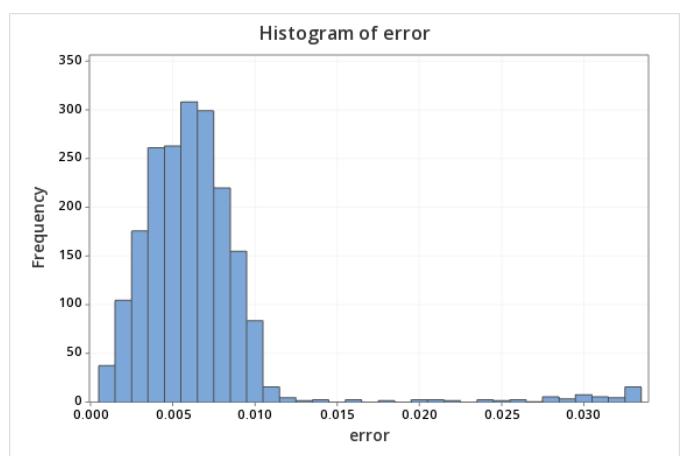


Figura 18: Distribución estadística de desempeño durante seguimiento autónomo

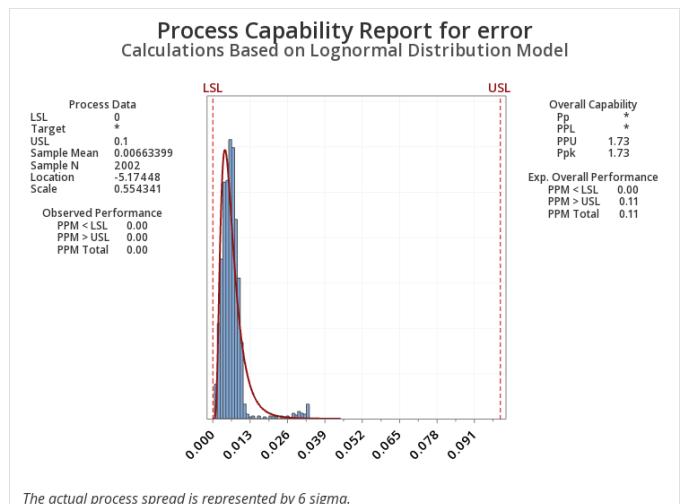


Figura 19: Prueba de capacidad no-normal para desempeño de conducción autónoma

A grandes rasgos, al tomar en cuenta lo presentado en las figuras 10, 17, 18, 19, se observa el error acumulado

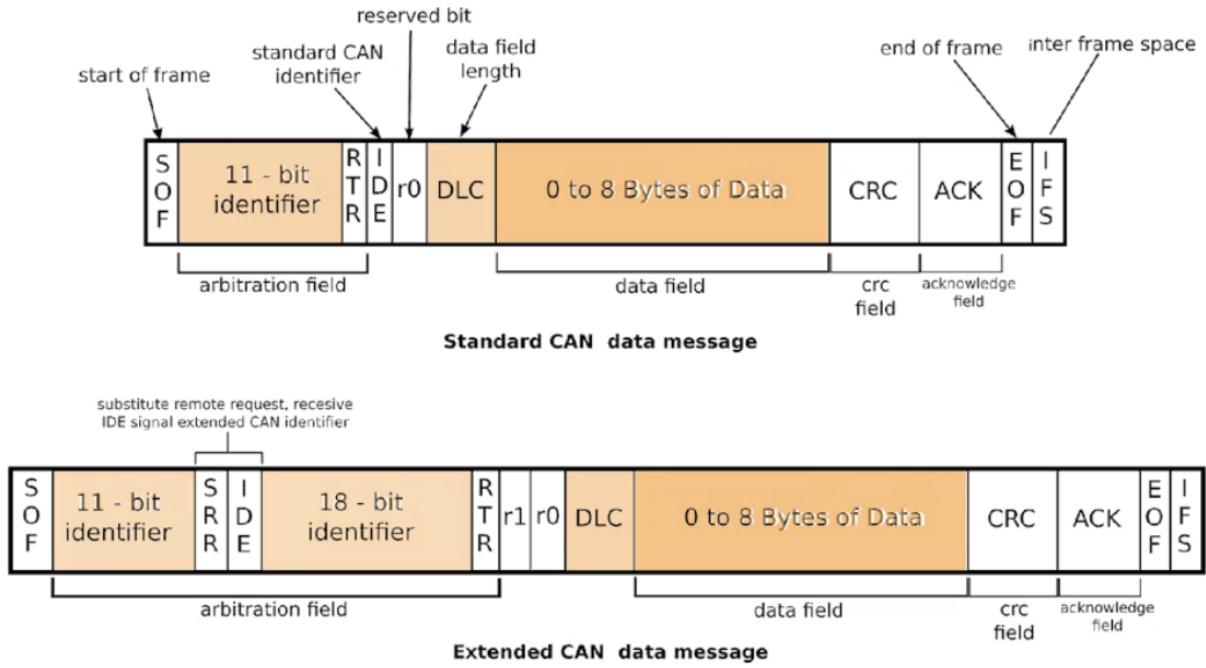


Figura 20: Esquema de la trama de mensajes en el protocolo CAN Bus.

por parte del controlador. Aunque sutil, su existencia afecta considerablemente durante la realización de la conducción autónoma. Aún cuando existen variaciones y aumentos del error, las pruebas actuales no sugieren un impacto significativo, véase la información obtenida del proceso de capacidad; la realidad es que, bajo las limitaciones, el modelo actual sugiere un proceso de control robusto, sobre restricciones "severas" dentro de los alcances obligados del propio proceso de navegación; i.e. el desarrollo podría ser comparable a otros proyectos de conducción autónoma por parte de empresas más grandes como lo pueden ser Google y Tesla [9], en su respectivo momento.

#### IV. ARQUITECTURA DEL SISTEMA TAMAÑO REAL (AMR1)

Esta sección detalla la arquitectura de software y la lógica operativa que gobiernan el prototipo del Autonomous Mobile Robot (AMR). A diferencia de las plataformas robóticas educativas que suelen centralizar el procesamiento en una única computadora embebida, el AMR1 implementa una arquitectura distribuida robusta basada en el estándar industrial **CAN Bus** (Controller Area Network 20) [10].

El diseño del sistema se fundamenta en un paradigma descentralizado. Se evita la dependencia de una unidad de procesamiento monolítica ("Maestro-Esclavo"), optando por la distribución de funciones críticas —frenado, dirección, tracción y control de mando— en nodos computacionales independientes [11].

Cada nodo consiste en un microcontrolador dedicado que ejecuta una tarea específica en tiempo real y se interconecta

con el resto del sistema a través de un bus diferencial de dos hilos. Esta arquitectura modular ofrece ventajas significativas en términos de mantenimiento y escalabilidad: permite la modificación, actualización o sustitución de un subsistema completo (por ejemplo, cambiar el algoritmo de control de dirección) sin alterar el código fuente de la tracción o el frenado, garantizando así la extensibilidad del proyecto a largo plazo.

##### IV-A. Implementación

**IV-A1. Protocolo de comunicación CAN bus:** El CAN bus es el eje central de la arquitectura de control del AMR1, ya que ofrece una capa de comunicación sólida y jerárquica (véase en 21 [11]).

- Estructura de comunicación

**Inicialización:** La primera línea del código incluye las bibliotecas necesarias (`mcp_can.h` y `SPI.h`). El puerto Serial se inicia y se procura establecer el controlador MCP2515 con una velocidad de baudios de 500 kbps. El sistema se para (`while(1)`) si la inicialización no tiene éxito, garantizando así que el programa funcione únicamente con un bus CAN que esté en funcionamiento.

**Transmisión (enviarMensajeCAN):** Gracias a esta función, el usuario puede mandar instrucciones CAN directamente desde el Monitor Serial (por ejemplo, 200 75). Divide la cadena de entrada en un valor entero (rango entre 0 y 100) y un identificador hexadecimal (ID). El valor se empaqueta en un solo byte y se envía mediante `CAN0.sendMsgBuf()`.

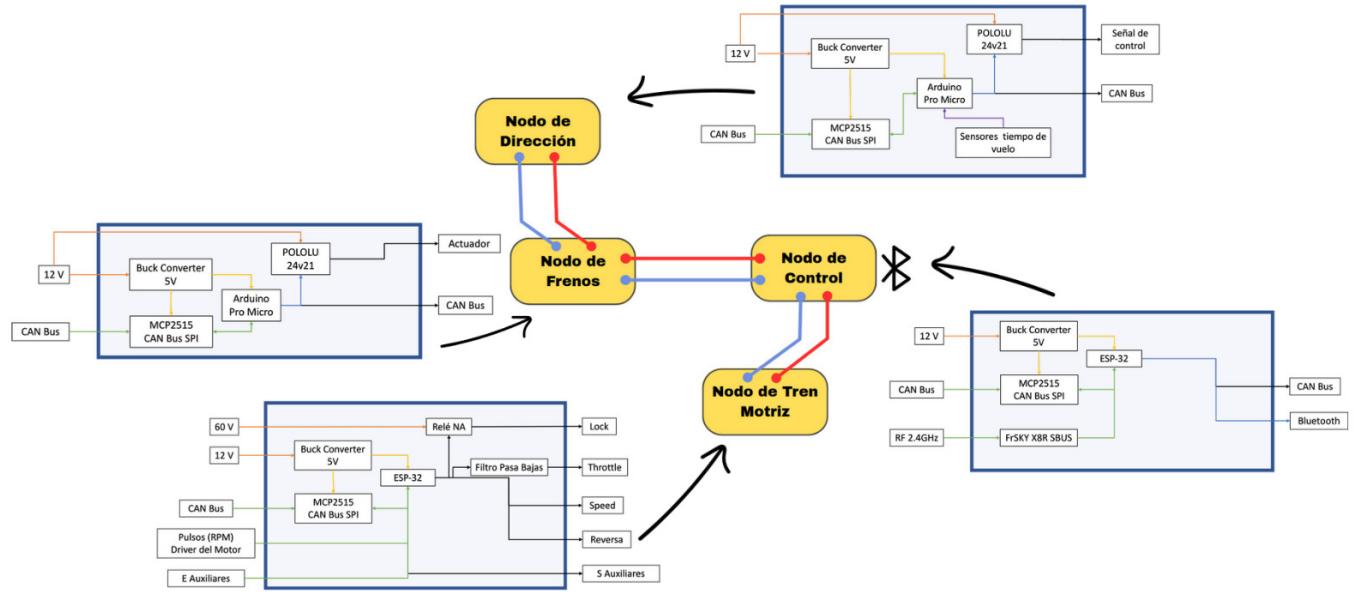


Figura 21: Sistema de comunicaciones y operación del AMR

Cuadro I: Mapa de Comunicación y Asignación de Tópicos en el Bus CAN

Módulo	Funcióñ	ID	Dirección	Rango	descripción
Global	Paro de emergencia	0x210	TX/RX	0 / 1	Comando de seguridad: Activa (1), desactiva (0).
Dirección	Comando de posición	0x100	TX	0–100 %	Posición objetivo del actuador (Centro: 51%).
Dirección	Feedback de posición	0x101	RX	0–100 %	Posición actual medida por potenciómetro.
Frenos	Comando de frenos	0x200	TX	0–100 %	Posición objetivo del actuador de freno.
Frenos	Feedback de frenos	0x201	RX	0–100 %	Posición actual medida por potenciómetro.
Tren motriz	Comando de potencia	0x300	TX	0–100 %	Velocidad (PWM) del motor.
Tren motriz	Feedback de potencia	0x301	RX	0–100 %	Potencia aplicada actual.
Tren motriz	Comando de relé	0x310	TX	0 / 1	Estado del relé: OFF (0) / ON (1).
Tren motriz	Feedback de relé	0x311	RX	0 / 1	Estado actual del relé.
Tren motriz	Comando de dirección	0x320	TX	0 / 1	Dirección de avance: Reversa (0) / Adelante (1).
Tren motriz	Feedback de dirección	0x321	RX	0 / 1	Dirección actual del avance.

**Recepción (leerCAN):** Esta función es crucial para el diagnóstico. Revisa de manera continua la recepción de mensajes e imprime los que se centran en las respuestas de los actuadores: Retroalimentación proporcional (0x101, 0x201, 0x301): Imprime el valor obtenido en forma porcentual (entre 0 y 100 %).

**Feedback binario (0x311):** Para el estado del relé, emplea un umbral ( $THRESH\_311 = 50$ ) que convierte el valor recibido (si se encuentra en porcentaje) a un estado binario de encendido/apagado (1/0).

#### ■ Mapa de identificadores

El sistema utiliza el siguiente conjunto de IDs Estándar para la orquestación de comandos y la recepción de telemetría. Cada mensaje utiliza un byte de datos (observe tabla I).

**IV-A2. Subsistemas actuadores:** Cada subsistema funciona de manera independiente, aplicando lógica de seguridad local y lazo de control.

#### 1. Jerarquía de seguridad (paro de emergencia)

La lógica de control de la dirección es jerárquica. El mensaje de Paro de Emergencia (0x210) es el que tiene la máxima prioridad:

- Si se recibe 0x210 con valor 1 (paroEmergenciaActivo = true), el código establece posicionObjetivo en 51 (la posición central del actuador) y omite la ejecución de cualquier comando 0x100.

- El bucle loop() garantiza que, mientras sea true paroEmergenciaActivo, la dirección se mantenga en el centro (posicionObjetivo = 51), dando prioridad a la estabilidad y al cero de dirección sobre

cualquier otro comando.

## 2. Estructura del nodo de dirección

El código de dirección controla la posición de un actuador lineal (servo), que es el que altera el ángulo de las ruedas. El control de la dirección se fundamenta en un ciclo ininterrumpido de ejecución:

Definición > Lectura Can > Estado Actual  
Estado actual > Feedback CAN > Actuador

**Descripción de los pines:** Configura los pines (`PIN_PWM`, `PIN_DIR`, `PIN_FEEDBACK`) y las especificaciones del bus CAN.

**Consecución del objetivo:** Para determinar la variable `posicionObjetivo`, el sistema espera la última instrucción de posición (0–100%) que le llegue a través del ID 0x100.

**Estado Actual y Filtrado:** La función "leerPotFiltrado()".aplica un filtro de media móvil (N=6) a las lecturas del potenciómetro con el fin de estabilizar la señal de retroalimentación. `lecturaToPorcentaje()` convierte estos valores en bruto (`RAW_MIN` a `RAW_MAX`) en un porcentaje (0–100%).

**Feedback:** La función `enviarFeedbackCAN()` envía al bus la posición del actuador en tiempo real (ID 0x201 — Nota: Hay una incongruencia en el código fuente, porque el mapa de IDs indica que para Dirección se debe usar 0x101).

## 3. Mecanismo de actuación

Se implementa un control de posición proporcional (P). Que determina el error entre la posición actual y la posición objetivo [12].

Si `error < Tolerancia`, se apaga el PWM (`analogWrite(PIN_PWM, 0)`). Cuando hay un error, el `PIN_DIR` determina la dirección de giro y el valor PWM se asigna directamente al valor absoluto del error. Esto provoca que la velocidad de corrección aumente en la medida en que el error también lo hace.

## 4. Estructura del nodo de frenos

El código de frenos controla un actuador con una lógica y un propósito parecidos a los del de Dirección, aunque centrado en la implementación de la fuerza de frenado.

**Comando (RX):** La posición objetivo (0–100%) se obtiene mediante el ID 0x200.

**Feedback (TX):** Evalúa la posición actual, utiliza un filtro de media móvil (N=5) y transmite el valor (0–100%) por medio del ID 0x201.

**Lógica de control:** Al igual que la Dirección, hace uso de un control proporcional para trasladar el actuador hasta la posición objetivo a una velocidad que depende

del error.

## 5. Estructura del nodo de tren motriz

**Comandos múltiples:** Este módulo tiene la capacidad de escuchar tres identificadores diferentes de comando: 0x300 (potencia), 0x310 (relé) y 0x320 (dirección de avance/reversa).

**Retroalimentación:** Para cada uno de sus parámetros (0x301, 0x311, 0x321), envía información de estado.

**Control de la potencia:** La función `rampaPWM()` escribe en el motor un valor PWM (pwmMin a pwmMax) que se corresponde con la variable de velocidad (0–100%).

**Filtros de la red CAN:** El `setup()` de este módulo establece filtros de hardware en el MCP2515 para que acepte, con exactitud, los mensajes de comando (0x300, 0x310, 0x320, 0x210), mejorando así el procesamiento de mensajes.

*IV-A3. Estructura del nodo de control:* El módulo de control es la interfaz entre el operador y el bus CAN, que convierte las señales de un mando a distancia SBUS en órdenes para la red.

La librería `sbus.h` es utilizada por el código para decodificar las señales del receptor. La lógica se lleva a cabo a un ritmo estable (cada 20 ms, aproximadamente 50 Hz).

- **Paro/Relé (Canal 4):** Este canal administra dos estados cruciales en un solo interruptor.
- **Palanca hacia abajo:** Habilita el Paro de Emergencia (0x210 con valor 1).
- **Palanca hacia arriba:** Desactiva el Paro (0x210 en valor 0) y activa el Relé (0x310 en valor 1).
- **Selección de modo (Canal 5):** Posibilita que el operador pase del Modo SBUS (control remoto activo) al Modo Serial (control manual para depuración).
- **Modo Lento/Rápido (Canal 6):** Modifica el límite superior de velocidad (maxVelocidad) de manera dinámica, lo que posibilita un manejo más seguro a velocidades bajas.
- **Rampa de Velocidad (Canal 0, Arriba):** Con el objetivo de prevenir la tensión mecánica en el tren motriz, no se implementa la velocidad de manera instantánea. Se estima una velocidad objetivo y se modifica progresivamente la velocidad actual (`velStep`) en cada ciclo, generando así un efecto de rampa de aceleración o desaceleración suave. El ID 0x300 se usa para enviar el valor final (`valorVel`).
- **Frenado (Canal 0, hacia abajo):** El joystick se mueve hacia abajo y se asigna a `valorFreno` (0-179). Para que una señal de 0% represente un Freno Mínimo (Actuador Abierto) y 179% sea igual a un Freno Máximo (Actuador Cerrado), se invierte el valor enviado al ID 0x200, es decir, se calcula como (179 - `valorFreno`).
- **Filtrado para la transmisión** El módulo de control se vale de zonas muertas para el envío a fin de disminuir el tráfico en el bus:

`A.abs(frenoEnviar - lastFreno)`

```

> DEADZONE_FRENO|
B.abs(valorDirCoche - lastDir)
> DEADZONE_DIR_ENVIO|

```

Esto garantiza que los mensajes se transmitan por el bus CAN únicamente si la variación en la variable es importante.

Al comienzo del programa, se aconseja seguir el protocolo de encendido siguiente:

- Encender el control remoto (TX).
- Cuando se empiece el programa, primero energizar el sistema a 12 V.
- Posteriormente, elevar el voltaje a 24 V. Este método permite que los subsistemas se inicien de manera rápida y segura. En cambio, se puede comenzar directamente a 24 V, aunque esto podría causar un desconectado temporal de los microcontroladores.
- Finalmente, adjuntar el voltaje final de 48 V, para desbloquear de manera completa las funciones de movimiento del vehículo.

#### IV-B. Validación Operativa del AMR1

La validación de la arquitectura electrónica y mecánica del AMR1 se realizó mediante pruebas de campo en exteriores, utilizando un esquema de teleoperación. Actualmente, los comandos de control se transmiten vía serial a través de un enlace Bluetooth.

Es crucial destacar que esta interfaz serial valida la capacidad del sistema para integrar un ordenador a bordo en el futuro; desde la perspectiva de la red CAN, es indiferente si los comandos provienen de un operador remoto o de un algoritmo de navegación autónoma, siempre que se respete el protocolo de comunicación establecido.



Figura 22: Pruebas de campo: Navegación teleoperada del AMR en entorno controlado.

Como se evidencia en las Figuras 22, 23 y 24, la integración de los nodos distribuidos permitió una navegación continua y funcional. Durante las pruebas se identificaron restricciones dinámicas, específicamente en el radio de giro y la velocidad máxima operativa.

Sin embargo, al considerar que se trata de un prototipo experimental manufacturado ad-hoc y no de un vehículo de producción comercial (OEM), el desempeño mecánico y la respuesta de la red CAN se consideran satisfactorios. La plataforma cumple con su propósito fundamental: servir como



Figura 23: Validación de la tracción y dirección bajo carga de pasajeros.

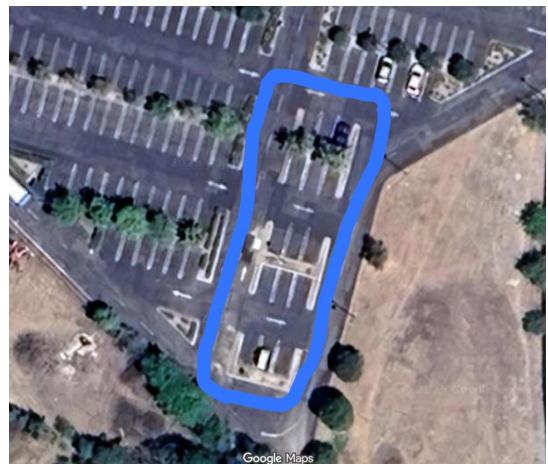


Figura 24: Traza del recorrido de prueba (Vista satelital).

un banco de pruebas robusto y modular, capaz de soportar la futura integración de los sistemas de autonomía validados en el QCar.

## V. FUSIÓN EXPERIMENTAL

### V-A. Entorno de Desarrollo y Ejecución

La implementación experimental del sistema autónomo en el AMR1 se desarrolló principalmente en una laptop **ASUS ROG Strix G16 (Intel Core i9, 32 GB RAM, Ubuntu 22.04, ROS 2 Humble)**, donde se ejecutaron todos los nodos de percepción, fusión sensorial y control en tiempo real.

Como plataforma de apoyo, se utilizó una laptop ASUS TUF Gaming (Ryzen, 16\,GB RAM, Windows~11 con WSL2 Ubuntu~22.04) para la ejecución secundaria de nodos de control, análisis de datos y depuración durante las pruebas experimentales.

Para la capa de adquisición sensorial del vehículo, se empleó un microcontrolador ESP32, encargado de la lectura de encoders, IMU y de la comunicación serial con los nodos ROS 2 del AMR1.

Adicionalmente, un sistema embebido NVIDIA Jetson Orin Nano fue utilizado en pruebas independientes realizadas en la plataforma AMR1, donde ejecutó nodos ROS 2 para adquisición de sensores y control en tiempo real.

## V-B. Desarrollo

Como etapa final de desarrollo (cierre de compartimentación) se realizó la fusión de ambos proyectos, en pos del desarrollo de una plataforma tamaño real capaz de ejecutar el seguimiento de una trayectoria real.

De acuerdo con la estructuración de ambos proyectos, el acople es relativamente simple al crear un nodo de ROS 2 que permita la traducción entre ROS y la comunicación serial del AMR1. Esa es la unión principal; sin embargo, al momento de realizar la fusión, es cuando existen ciertas áreas adaptables dentro del propio desarrollo. Primeramente, está el equipamiento de la plataforma 1:1, debido a la ausencia de algunos sensores mediante los cuales opera la parte técnica (se incorporó un IMU similar al del Qcar, y un encoder especializado para motores eléctricos). Prosiguiendo con las adaptaciones, se montó un sistema de conectividad (inclusión de un ordenador a bordo, conectado con un router móvil), de modo que el AMR1 funcionara de modo similar al Qcar, en términos técnicos. Finalmente, la adaptación de los mensajes enviados desde ROS por Pyserial; exceptuando la velocidad, todos los comandos funcionaban de manera óptima al trasladarse al AMR1; con respecto a la velocidad, se debía publicar dos veces al momento de ser enviado a serial, esto podría deberse a la programación entre el control remoto original, y el AMR1, sin embargo, no existe evidencia que soporte esta teoría.

Se adjuntan evidencias del proceso de navegación autónoma.



Figura 25: Pruebas de campo: Navegación autónoma del AMR1 en entorno controlado.



Figura 26: Pruebas de campo: Navegación autónoma del AMR1 en entorno controlado.



Figura 27: Pruebas de campo: Navegación autónoma del AMR1 en entorno controlado.

Al observar el comportamiento del AMR1 durante la navegación autónoma, primeramente se puede considerar un acercamiento exitoso en la conversión de escalas mediante la fusión de proyectos. Ciertamente, el error fue aumentando sustancialmente durante los instantes de tiempo del proceso correspondiente; sin embargo, esto se atribuye a diferentes características que afectan el desempeño del controlador; tómese el diagrama de Ishikawa mencionado previamente, en el cuál sugería que el vehículo a implementar tenía mucho que ver en los resultados de control, efectivamente al realizar el manejo desde la plataforma 1:1 se observan como los impactos de las variables como la caracterización, físicas internas del automóvil, entre otras, realmente hacen un cambio al hablar del seguimiento de trayectorias realizado. Con respecto al ambiente, las pruebas son contrarias, debido a que aún cuando se poseía un sitio de pruebas relativamente de menores adecuaciones, la realidad es que el AMR1, logró superar las pruebas de manera satisfactoria.

## VI. DISCUSIÓN Y TRABAJO FUTURO

La ejecución paralela de la validación algorítmica en el QCar y la rehabilitación del AMR1 ha permitido contrastar la lógica de control frente a las restricciones físicas de hardware. A diferencia de un enfoque puramente teórico, la integración experimental ("Fusión Experimental") reveló desafíos no lineales en la interfaz de comunicación que no eran evidentes en la simulación.

### VI-A. Análisis del Control y Variabilidad (QCar)

Si bien el algoritmo *pure pursuit* demostró ser eficaz para el seguimiento de trayectorias con curvatura moderada, el análisis de los resultados sugiere que el error no es únicamente producto de la geometría del algoritmo (corte de curvas por *lookahead*).

Mediante la aplicación de herramientas de calidad (Diagrama de Ishikawa) 16, se detectó cualitativamente que una parte sustancial de la varianza en el seguimiento de trayectoria se origina en factores ambientales y físicos del vehículo, más que en el código en sí. A pesar de mantener entradas idénticas de configuración ( $k = 0,4$ ,  $v_{ref} = 0,05$ ,  $Look-ahead - distance = 0,15$ ), se observaron variaciones en el esfuerzo de control en distintas pruebas. Esto indica que para alcanzar una autonomía robusta, es necesario migrar

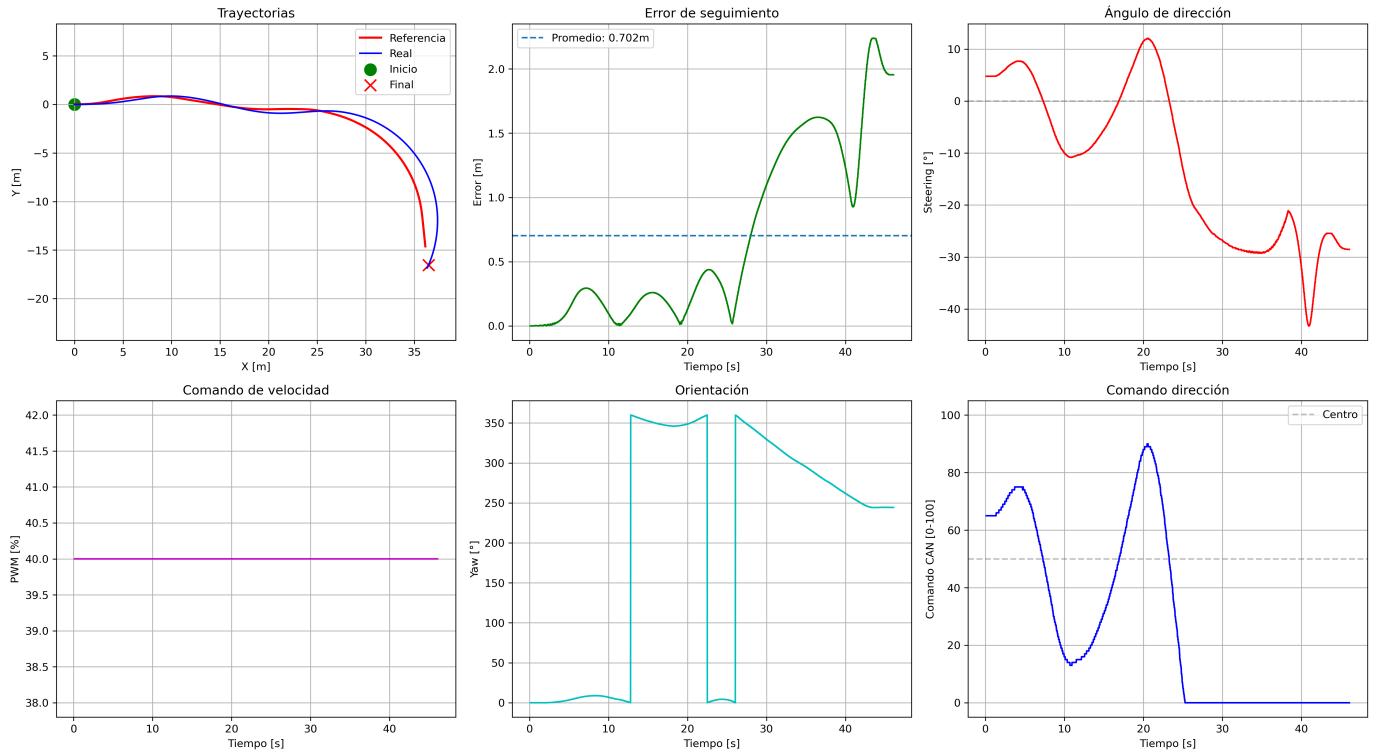


Figura 28: Desempeño de navegación autónoma del AMR.

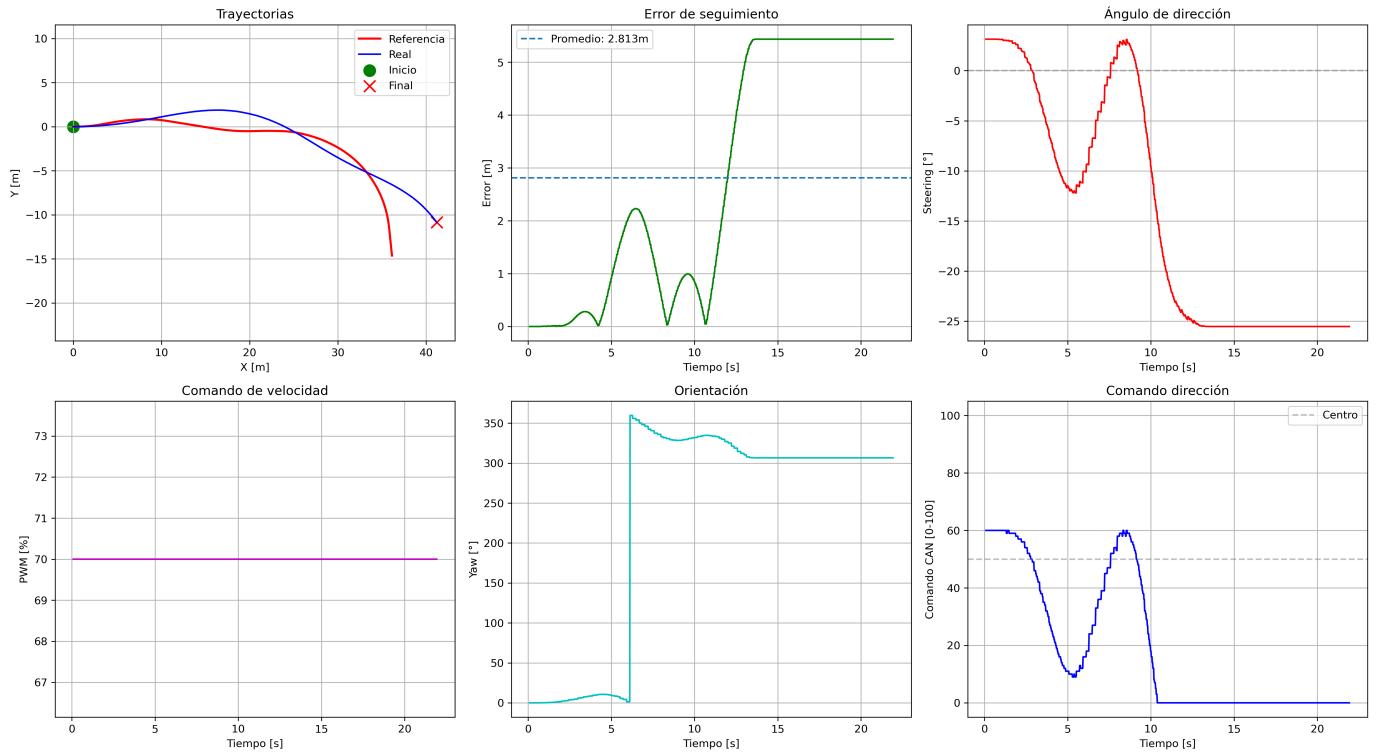


Figura 29: Desempeño de navegación autónoma del AMR.

de controladores geométricos a estrategias que modelen la dinámica del vehículo:

- **Robustez ante Perturbaciones:** Se recomienda implementar controladores como *Stanley* [13] o MPC [14], los cuales pueden compensar mejor las no-linealidades (fricción, inercia) que el diagrama de Ishikawa identificó como fuentes de error [6].

#### VI-B. Resultados de la Arquitectura Distribuida y Fusión

La validación del AMR1 confirmó la superioridad de la arquitectura descentralizada en CAN Bus para la seguridad intrínseca. Sin embargo, la fusión experimental arrojó hallazgos críticos sobre la interfaz ROS-Serial:

1. **Latencia y Redundancia:** Durante la traducción de comandos de ROS 2 a trama serial, se identificó que el comando de velocidad requería ser publicado con redundancia (doble envío) para ser interpretado correctamente por el microcontrolador de tracción. Esta peculiaridad sugiere la necesidad de optimizar la sincronización de hilos en el nodo puente (*bridge node*).
2. **Escalabilidad Exitosa:** A pesar de las diferencias mecánicas, la lógica de navegación validada en el QCar (escala 1:10) logró operar la plataforma AMR1 (tamaño real) de manera funcional, validando la hipótesis de que la arquitectura de software desarrollada es adaptable a la escala del vehículo.

## VII. CONCLUSIÓN

El presente trabajo ha culminado exitosamente con la validación de una metodología integral para el desarrollo de vehículos autónomos, logrando el cierre de la brecha entre la simulación algorítmica y la implementación física. La segmentación estratégica del proyecto permitió utilizar el QCar como un entorno seguro para la iteración rápida de algoritmos de fusión sensorial (EKF) y control, mientras que la rehabilitación del AMR1 estableció una base mecatrónica robusta y segura mediante el protocolo CAN Bus.

La fase final de fusión experimental demostró que la arquitectura de software basada en ROS 2 es escalable y portable. Aunque se identificaron limitaciones dinámicas en los controladores geométricos y desafíos de sincronización en la interfaz de comunicación serial, el sistema logró ejecutar navegación autónoma en una plataforma de escala completa.

En consecuencia, este proyecto entrega no solo un prototipo funcional, sino un marco de referencia académico validado ("banco de pruebas") que deja el camino despejado para la investigación avanzada en percepción, planificación de movimiento y control predictivo en escenarios de movilidad real.

## APÉNDICE A RECURSOS DEL PROYECTO

#### A-A. Repositorio de Código

El repositorio completo del proyecto se encuentra disponible en: [https://github.com/abrahammorohdez19/smart\\_mobility\\_2025](https://github.com/abrahammorohdez19/smart_mobility_2025)

#### A-B. Video Demostrativo

Un video demostrativo de los sistemas ejecutándose puede consultarse en: <https://youtu.be/7VoVAsYSaZo?si=ZbUy-V-Ub0IkP2O>

## REFERENCIAS

- [1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.
- [2] H. G. G.-H. Jorge Antonio Reyes Avendaño, "Development of a simplified automated guided electric vehicle for testing and teaching automotive systems and navigation algorithms," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 1, no. 1, 2020.
- [3] D. O. Sánchez, "Sensor fusion algorithms for autonomous vehicles," Master's thesis, Instituto Tecnológico de Estudios Superiores de Monterrey, 2021.
- [4] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie Mellon University, The Robotics Institute, Tech. Rep. CMU-RI-TR-92-01, 1992.
- [5] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.
- [6] R. Rajamani, *Vehicle Dynamics and Control*, 2nd ed. Springer Science & Business Media, 2011.
- [7] S. Macenski, T. Foote, B. Gerkey, C. Clay, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.
- [8] *What is a Fishbone Diagram? Ishikawa cause and effect diagram*, 2025.
- [9] *Overall capability for Normal Capability Analysis for Multiple Variables*, 2025.
- [10] M. Di Natale, H. Zeng, P. Giusto, and A. Ghosal, *Understanding and Using the Controller Area Network Communication Protocol: Theory and Practice*. Springer Science & Business Media, 2012.
- [11] Bosch, "Can specification version 2.0. robert bosch gmbh," Bosch, Tech. Rep., 1991.
- [12] K. Ogata, *Modern Control Engineering*, 5th ed. Prentice Hall, 2010.
- [13] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.
- [14] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.