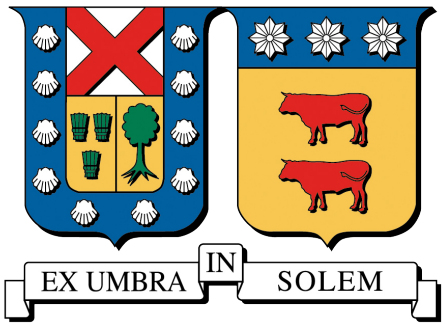
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAISO - CHILE



IMPLEMENTACION DE MEJORAS EN HARDWARE Y SOFTWARE DE PLATAFORMA RUPU PARA EXPERIMENTOS DE PELOTONES DE VEHÍCULOS SEGUIDORES DE LÍNEA

FELIPE RIFO MARDONES

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRÓNICO

PROFESOR GUÍA	:	DR. GONZALO CARVAJAL B.
PROFESOR CORREFERENTE	:	DR. FRANCISCO VARGAS P.
PROFESOR CORREFERENTE	:	DR. ANDRÉS PETERS R.

MARZO 2025

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: IMPLEMENTACIÓN DE MEJORAS EN HARDWARE Y SOFTWARE DE PLATAFORMA RUPU PARA EXPERIMENTOS DE PELOTONES DE VEHÍCULOS SEGUIDORES DE LÍNEA

Nombre del candidato(a): Felipe Rifo Mardones

Carrera / Grado: Ingeniería Civil Electrónica

Campus: Casa Central Valparaiso **;** **Departamento**: Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Gonzalo Carvajal, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 29-07-2025 ; **Firma:** _____

Estudiante o Candidato(a):

Fecha: 18-07-2025 ; **Firma:** _____

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Índice general

1. Introducción **5**
1.1. Motivación y contexto 6
1.2. Planteamiento del problema 7
1.3. Alcances y contribuciones 8
1.4. Organización del informe 9

2. Antecedentes **10**
2.1. Versión actual de plataforma RUPU 10
2.1.1. Características de los agentes móviles 11
2.2. Validación experimental de la plataforma 17
2.2.1. Experimento para evaluar desempeño de sensor de distancia VL53L0X . 17
2.2.2. Experimento para evaluar repetibilidad - arreglo QTR 19
2.3. Mejoras propuestas 22

3. Implementación de mejoras **25**
3.1. Compatibilidad en el protocolo de comunicación 25
3.1.1. Optimización en la adquisición de datos 26

4. Experimentos y resultados **33**
4.1. Metodología experimental 33

5. Conclusiones y trabajo futuro **39**
5.1. Conclusiones 39
5.2. Trabajo Futuro 40

6. Anexos **42**
6.1. Rediseño de la estructura de los agentes 42
6.1.1. Fuente de poder para la nueva versión de los agentes 43
6.1.2. Diseño de PCB 45
6.2. Pruebas de desempeño Python y C 46
6.3. Consideraciones sobre el uso de la nueva versión 48
6.3.1. Actualización de sistema operativo y migración de código para bibliotecas compatibles 48
6.3.2. Ejecución automatizada y parametrizada del script de Python 48

Índice de tablas

2.1. Comandos definidos para el operador de la plataforma experimental. 13
2.2. Resultados de RMSE según iteraciones para diferentes carros 22

3.1. Estadísticas de profiling relevantes en el ciclo de control 30
3.2. Estadísticas de profiling relevantes en el ciclo de control 32

4.1. Resultados RMSE por iteración para cada carro 35
4.2. Resultados RMSE por iteración para cada carro 38

6.1. Comparativa desempeño algoritmo de control en Python y C 47
6.2. Comandos de configuración inicial 49

Índice de figuras

2.1. Componentes principales de un agente móvil 12
2.2. Diagrama de bloques de conexión para agentes ESP32 13
2.3. versión Raspberry Pi de los agentes 14
2.4. Diagrama de bloques de conexión para agentes Raspberry Pi 15
2.5. Esquema de control implementado en RUPU 16
2.6. Diagrama de estados de operación de los agentes 16
2.7. Agente modificado para experimento 18
2.8. Mediciones de distancia registradas en experimento bajo diferentes condiciones ambientales 19
2.9. Pista utilizada para experimentos de repetibilidad 20
2.10. Medición de distancia en una vuelta de la pista utilizada 21
2.11. Segmentación de la pista y su relación con la distancia medida 22
2.12. Resultados experimento repetibilidad 23

3.1. Desempeño inicial versión Raspberry Pi 26
3.2. Estadísticas de profiling. 27
3.3. Estadísticas de profiling con ADC MCP3208 29
3.4. Tiempos de latencia ciclo de control 30
3.5. Tiempos de latencia ciclo de control optimizado 32

4.1. Configuraciones de pelotón heterogéneo agentes ESP32 y Raspberry Pi 34
4.2. Pista utilizada en experimentos de repetibilidad con pelotón heterogéneo 35
4.3. Resultados experimento repetibilidad con agente Raspberry Pi como líder . . . 36
4.4. Resultados experimento repetibilidad con agente Raspberry como 3er carro en pelotón mixto 37

6.1. Robot JetBot 43
6.2. Fuente de Poder UPS 3S 44
6.3. Agentes RUPU destacando la conexión por cable de sensores 45
6.4. Propuesta de PCB reemplazo conector 45
6.5. Versión final nueva versión de agente RUPU Raspberry Pi 46

El trabajo desarrollado en esta memoria de título fue parcialmente financiado por medio de los proyectos FONDECYT Regular 1241813 y FONDECYT Regular 1251406.

1 | Introducción

El presente trabajo contribuye al desarrollo y mejora de la plataforma experimental RUPU, diseñada en el Departamento de Electrónica para la evaluación de algoritmos de control distribuido en pelotones de vehículos autónomos. Esta plataforma está compuesta por diversos elementos, entre los cuales destacan los agentes móviles, responsables de ejecutar de forma distribuida las tareas de navegación y control. Si bien la plataforma, junto con la primera versión de estos agentes móviles, fue previ amente diseñada, fabricada y validada a nivel funcional [1, 2], su versión más reciente presenta limitaciones en términos de compatibilidad, usabilidad y repetibilidad de experimentos.

El trabajo desarrollado en esta memoria de título se centra en mejorar la infraestructura de los agentes de la plataforma, analizando los aspectos y retos que deben abordarse para lograrlo, con un enfoque particular en una nueva versión basada en Raspberry Pi, extendiendo las capacidades de los prototipos diseñados en trabajo previo [3] y asegurando su interoperabilidad con los distintos componentes que conforman la plataforma. Para ello, se realizaron mejoras en los protocolos de comunicación, se analizó el código encargado del control de los agentes mediante técnicas analíticas de profiling (perfilamiento de desempeño del código) y se aplicaron optimizaciones para mejorar su desempeño temporal. Además, se evaluaron alternativas para mejorar los algoritmos de adquisición de datos y se rediseñó la estructura física en la que se monta el hardware para una mejor distribución de las componentes y sus conexiones. Estas mejoras, junto con las pruebas de validación asociadas, buscan consolidar la versión basada en Raspberry Pi como un componente integral de RUPU, permitiendo la experimentación con algoritmos de control en un entorno escalable. De este modo, se espera que la plataforma pueda ser utilizada de manera confiable, facilitando la realización de experimentos repetibles y reproducibles para futuras investigaciones sobre conducción autónoma y control cooperativo.

En este capítulo se presenta el contexto y la motivación del estudio, destacando la importancia de la plataforma RUPU en la validación de algoritmos de control distribuido. Se describe el problema que se busca abordar, así como los objetivos y alcances del trabajo. Además, se detallan las principales contribuciones de esta investigación y la estructura del documento, proporcionando una visión general del desarrollo y la organización de los capítulos siguientes.

1.1. Motivación y contexto

El avance en las tecnologías de conducción autónoma ha impulsado el estudio de sistemas multi-agente (MAS) aplicados a pelotones de vehículos autónomos, los cuales presentan desafíos en control distribuido, comunicación y sincronización. Una de sus aplicaciones clave es la formación en pelotón, donde varios vehículos siguen a un líder manteniendo una distancia configurable, con potencial para el desarrollo de sistemas de transporte inteligente (ITS) [4] y aplicaciones como Cooperative Adaptive Cruise Control (CACC) [5] y Sistemas de Asistencia a la Conducción (ADAS) [6]. El estudio de pelotones ofrece múltiples beneficios para estas aplicaciones, como eficiencia energética, reducción de emisiones, mejora de la seguridad vial y optimización del tráfico [7–9]. Sin embargo, existen desafíos técnicos que afectan su desempeño, como la estabilidad de cuerda [10], fenómeno en el cual pequeñas variaciones en la velocidad de un vehículo se propagan y amplifican a lo largo del pelotón, comprometiendo su cohesión, especialmente en configuraciones grandes o con latencias en la comunicación. Otros retos incluyen la gestión de retrasos en la comunicación, que puede desincronizar el movimiento coordinado, y la robustez ante fallos en sensores, esencial para mantener la seguridad en condiciones adversas.

Desde la perspectiva del control, la formación en pelotón representa un problema dual [11]: el control lateral, que mantiene al vehículo en el camino predefinido, y el control longitudinal, que asegura una distancia constante con el predecesor. Resolver ambos problemas de control de manera simultánea es desafiante debido a las dinámicas acopladas, impulsando la investigación en control multivariable para abordar estas interacciones de manera integrada [12–15]. Factores prácticos como el ruido en sensores, pérdida de paquetes en la comunicación e interferencias ambientales dificultan aún más el desempeño del pelotón, resaltando la importancia de diseñar algoritmos robustos capaces de adaptarse a condiciones variables y mantener la estabilidad del sistema incluso ante perturbaciones [16–19].

Validar algoritmos de control en condiciones de operación representativas de entornos prácticos resulta crucial, ya que las simulaciones no capturan todas las complejidades operacionales. Aunque se han realizado pruebas de formación de pelotones con vehículos a escala real, como por ejemplo camiones de carga [20], estas suelen enfocarse en demostraciones tecnológicas con fines comerciales, y su elevado costo limita su utilidad en contextos de investigación académica y en la exploración de nuevos algoritmos. Como alternativa para superar estas restricciones, se proponen plataformas a escala reducida que permiten replicar los desafíos del control lateral y longitudinal de forma accesible y con condiciones reproducibles. Si bien existen múltiples plataformas educacionales basadas en vehículos autónomos, incluidas algunas disponibles comercialmente [21], estas tienden a ser costosas o carecen de sensores adecuados para abordar simultáneamente ambos problemas de control.

En este contexto, la plataforma RUPU, desarrollada en la Universidad Técnica Federico Santa María [1, 22], se presenta como una herramienta modular y de bajo costo para la validación experimental de algoritmos de control distribuido en pelotones de vehículos. RUPU permite emular situaciones reales de formación en pelotón, facilitando la experimentación y evaluación en condiciones diversas. Su diseño proporciona un entorno seguro que ofrece cierto grado de reproducibilidad en entornos controlados [2], abordando las complejidades del control y la comunicación en sistemas multi-agente, consolidándose como una alternativa accesible para la investigación académica y docente.

En trabajos previos, se ha desarrollado y validado la versión de los agentes de la plataforma RUPU basada en el microcontrolador ESP32 como unidad de cómputo [1], demostrando su eficiencia en la ejecución de algoritmos de control distribuidos para formaciones de pelotón. Además, en esta versión se han desarrollado herramientas que buscan aumentar su reproducibilidad y usabilidad, incluyendo una GUI (*Graphic User Interface*) y documentación detallada sobre el armado y uso de la plataforma, así como experimentos para validar la repetibilidad [1] de los algoritmos implementados [2].

Posteriormente, se desarrolló una versión basada en Raspberry Pi [3] con el objetivo de integrar visión por computador a la plataforma. Actualmente, esta versión se encuentra en una etapa temprana de prototipo y fue concebida como una prueba de concepto, con el objetivo de implementar visión por computador a través de una Raspberry Pi, utilizando como base el mismo chasis de un agente con ESP32, lo que implica que su diseño no está optimizado para su nueva funcionalidad. La versión basada en Raspberry Pi se desarrolla con el propósito de ampliar las capacidades de la plataforma, permitiendo experimentar con nuevos algoritmos y tecnologías más avanzadas, así como integrar de forma más eficiente los componentes y protocolos necesarios para el control distribuido. Su estado de prototipo actual justifica la necesidad de realizar pruebas exhaustivas que midan su rendimiento, eficiencia y capacidad para reproducir resultados de manera confiable, asegurando así su viabilidad para futurs implementaciones en la plataforma.

Este trabajo de memoria propone una nueva versión de la plataforma RUPU, orientada a mejorar su desempeño operativo, facilitar la implementación de algoritmos de control y aumentar la repetibilidad de los experimentos. La propuesta contempla modificaciones estructurales en el chasis del vehículo e incorpora una arquitectura electrónica basada en una Raspberry Pi como unidad de procesamiento central. Estas mejoras buscan expandir las capacidades de la plataforma en contextos educacionales y de investigación, habilitando a futuro la evaluación de algoritmos de control que demanden mayor capacidad computacional y la integración de sensores adicionales.

1.2. Planteamiento del problema

A medida que la plataforma RUPU ha progresado, no solo se han desarrollado diferentes versiones con el objetivo de ampliar sus capacidades y validar su desempeño en distintos escenarios experimentales, sino que también se han implementado diversas herramientas y documentación para mejorar su usabilidad y reproducibilidad. Si bien se ha avanzado y trabajado en la implementación y validación de la versión basada en ESP32, la versión basada en Raspberry Pi aún enfrenta desafíos técnicos y estructurales que requieren ser abordados. Además, hasta ahora no se ha probado la incorporación de esta última versión en la plataforma RUPU con el fin de conformar pelotones heterogéneos, lo cual representa una oportunidad importante para futuros estudios y experimentos que exploren la integración y cooperación de distintas arquitecturas dentro del mismo sistema.

Asimismo, se tomará en cuenta la definición de "repetibilidad" establecida por la *Association for Computing Machinery* (ACM) [23], la cual establece que una medición debe poder obtenerse con la precisión indicada por el mismo equipo, utilizando el mismo procedimiento de medición,

[1] En el contexto de este trabajo, se usan los términos reproducibilidad y repetibilidad en base a la definición entregada por ACM [23]

el mismo sistema de medición y bajo las mismas condiciones operativas en múltiples ensayos. En el contexto de experimentos computacionales, esto significa que un investigador debe poder repetir de manera confiable sus propios cálculos. Considerando esta definición, las mejoras implementadas en la plataforma RUPU estarán orientadas a garantizar un entorno de experimentación donde la repetibilidad sea un criterio central, asegurando así la confiabilidad y validez de los resultados obtenidos.

Por otra parte, se analizan los factores que dificultan la integración y funcionalidad de la nueva versión en la plataforma, así como las mejoras necesarias para garantizar su estabilidad y eficiencia en el estudio de algoritmos de control distribuido.

Este trabajo se plantea como objetivo validar y optimizar la versión basada en Raspberry Pi, centrándose en los siguientes aspectos:

- Evaluación del rendimiento general, comparando su desempeño con la versión basada en ESP32.
- Validación de la interoperabilidad con los componentes actuales de la plataforma RUPU, logrando la implementación de un pelotón heterogéneo.
- Pruebas experimentales para evaluar la repetibilidad de la nueva versión de los agentes, su capacidad en el control distribuido y en la formación de pelotones.

Con este enfoque, se busca que la versión basada en Raspberry Pi no solo complemente, sino que se integre plenamente en la plataforma RUPU, consolidándose como una herramienta fundamental para la experimentación y validación de algoritmos de control cooperativo y sistemas multi-agente.

1.3. Alcances y contribuciones

Este trabajo se enmarca dentro del proceso de mejora y validación de la versión de la plataforma RUPU basada en Raspberry Pi, con énfasis en la evaluación de su funcionalidad dentro de pelotones heterogéneos de vehículos autónomos usando distintos procesadores centrales. A continuación, se detallan los principales alcances y contribuciones de esta investigación:

- Debido a restricciones de recursos y tiempo, el desarrollo e implementación se ha realizado utilizando un único agente basado en Raspberry Pi. No obstante, este prototipo permitió validar su integración en pelotones compuestos por agentes heterogéneos, demostrando su interoperabilidad con agentes previamente desarrollados bajo arquitectura ESP32.
- El estudio se ha centrado en la validación de la *repetibilidad* de los experimentos, entendida como la capacidad de obtener resultados consistentes bajo las mismas condiciones operativas y el mismo equipo de colaboradores. En este trabajo, la repetibilidad se evalúa a partir de la consistencia de la distancia entre vehículos a lo largo de múltiples repeticiones de un mismo experimento, ya que esta variable representa de forma directa el desempeño del control longitudinal. Por otro lado, el control lateral tiene un mayor impacto en la estabilidad del pelotón, y las inconsistencias en su funcionamiento pueden ser identificadas visualmente durante la ejecución. No se abordan formalmente criterios de *reproducibilidad* ni *replicabilidad* [23], ya que dichas propiedades requieren condiciones adicionales de validación que exceden el alcance temporal y experimental de este trabajo.

- Finalmente, este trabajo no contempla el estudio ni la evaluación de distintos esquemas de control, ya que el objetivo principal es validar la funcionalidad e integración de los agentes en pelotones heterogéneos, utilizando el esquema de control base ya implementado en versiones anteriores de la plataforma.

Entre las contribuciones de este trabajo, se destacan las mejoras significativas implementadas en el software de la plataforma, tales como la optimización de los algoritmos de adquisición de datos, la reducción de la latencia en el ciclo de control y la incorporación de un esquema de comunicación compatible con el resto del sistema. Estas mejoras fueron validadas mediante pruebas experimentales en pista, con especial énfasis en la evaluación de la repetibilidad del sistema. Aunque se propusieron mejoras de hardware (incluyendo un rediseño estructural del chasis y la incorporación de una nueva fuente de poder), estas no fueron consideradas críticas para el estudio de la repetibilidad y, por lo tanto, no se evaluaron experimentalmente; sin embargo, se documentaron en anexos para facilitar su implementación futura.

Finalmente, todo el código fuente, scripts experimentales y documentación asociada se encuentra disponible en un repositorio público de GitHub [24], lo que refuerza el compromiso con la transparencia y fomenta la reutilización del trabajo en investigaciones posteriores.

1.4. Organización del informe

El resto de este informe se organiza como se describe a continuación:

- **Capítulo 2 - Antecedentes**: Se presenta un análisis del estado actual de la plataforma RUPU y su evolución a lo largo de trabajos previos. Además, se detallan los experimentos realizados para identificar fuentes de error que afectaron estudios anteriores y para validar la repetibilidad de la plataforma en su estado actual. También, se describen las características técnicas de la versión basada en ESP32 y la versión prototipo en Raspberry Pi, destacando sus ventajas y limitaciones.
- **Capítulo 3 - Implementación de mejoras**: Primero, se realizó un análisis estadístico para identificar posibles cuellos de botella en los algoritmos de adquisición de datos. Posteriormente, se detallan las optimizaciones realizadas en la plataforma, incluyendo la integración de una nueva versión basada en Raspberry Pi, la optimización de algoritmos de adquisición de datos y la mejora del protocolo de comunicación que mejoraron su desempeño y la eficiencia en la ejecución de los experimentos.
- **Capítulo 4 - Experimentos y resultados**: Se presentan los experimentos realizados para evaluar la estabilidad y el desempeño de la plataforma, centrándose en la repetibilidad de los resultados y la interoperabilidad entre agentes ESP32 y Raspberry Pi. Se analizan métricas clave que validan el rendimiento de las mejoras implementadas y se discuten las observaciones derivadas de las pruebas experimentales.
- **Capítulo 5 - Conclusiones y trabajo futuro**: Se resumen los principales hallazgos de este trabajo, destacando el impacto de las optimizaciones realizadas y su contribución al desarrollo de la plataforma RUPU. Además, se identifican oportunidades de mejora y se proponen líneas de trabajo a futuro para continuar con la evolución del sistema, incluyendo la integración de nuevas tecnologías y la optimización de su eficiencia energética.

2 | Antecedentes

Como parte del trabajo desarrollado previamente en la asignatura Proyecto de Título (ELO307), se realizó una evaluación experimental preliminar de la plataforma disponible al momento de comenzar este trabajo, apuntando a reproducir resultados previos e identificar posibles fuentes de error e incerteza en los experimentos previamente reportados. Adicionalmente, se presentan las características específicas de los agentes que utilizan Raspberry Pi como unidad de cómputo, junto con los problemas asociados a su funcionamiento. A partir de estos análisis, se proponen mejoras enfocadas en mejorar la usabilidad de la plataforma y aumentar la repetibilidad de los experimentos.

2.1. Versión actual de plataforma RUPU

La plataforma RUPU ha sido diseñada como una herramienta experimental para el estudio de algoritmos de control en formaciones de pelotón, abordando simultáneamente los problemas de control lateral y longitudinal. Su diseño modular y de bajo costo, junto con su capacidad de extenderse a áreas como el reconocimiento de imágenes para la detección de caminos, la convierten en una solución innovadora, versátil y accesible enfocada en configuraciones de pelotón.

Al momento de comenzar este trabajo de título, la versión más reciente de la plataforma se compone de diferentes elementos para ejecutar los experimentos. Uno de los elementos principales corresponde a los agentes móviles, vehículos autónomos que poseen capacidad de cómputo para implementar, junto a diferentes sensores, algoritmos de comunicación y control para resolver los problemas de seguimiento de carril y de distancia. Estos agentes incorporan, además, comunicación inalámbrica para compartir información entre sí, así como para recibir y enviar instrucciones y mediciones a un monitor/operador.

El monitor/operador, a través de una interfaz de usuario desarrollada localmente, envía instrucciones a cada agente para configurar el pelotón y definir el modo de operación de cada uno. Además, recibe y almacena mediciones de los agentes para su posterior procesamiento y análisis.

Por otra parte, un elemento esencial para la ejecución de los experimentos es la pista sobre la que operan los agentes. En este caso, consiste en una pista de goma EVA formada por tapetes negros de $29 \times 29 \times 0.8$ cm, marcados con una línea blanca de 2 cm que indica el recorrido, el cual puede incluir secciones rectas y curvas con un radio de 12.5 cm. Este diseño modular facilita el armado y la reconfiguración de la pista para adaptarse a distintos escenarios de prueba.

2.1.1. Características de los agentes móviles

El análisis de las características de los agentes móviles resulta fundamental para comprender su estructura y el alcance de sus aplicaciones en experimentación y validación de algoritmos de control. Inicialmente, se desarrolló una versión basada en ESP32, la cual permitió validar los algoritmos de control y comunicación en un entorno experimental controlado. Posteriormente, con el objetivo de expandir las capacidades de la plataforma, se diseñó una versión basada en Raspberry Pi, enfocada en la integración de algoritmos más avanzados, incluyendo visión por computador. En esta sección se presenta un resumen general de los aspectos mecánicos, electrónicos y algorítmicos relevantes para este trabajo. Los detalles completos del diseño e implementación de ambas versiones se encuentran disponibles en trabajos previos [1, 3]. El análisis se centra en los elementos que influyen en los objetivos de repetibilidad definidos por la ACM [23], así como en la confiabilidad de los resultados y la eficiencia en la configuración de pelotones.

Características de los agentes basados en ESP32

Los agentes móviles basados en ESP32 fueron la primera versión desarrollada en la plataforma RUPU, permitiendo establecer una base funcional para la experimentación en control distribuido. Esta versión ha alcanzado un nivel de desarrollo avanzado, lo que ha posibilitado su aplicación en múltiples experimentos, en los cuales se ha validado su desempeño tanto en pruebas individuales como en configuraciones de pelotones operando en entornos controlados [1, 2]. Además, estos agentes han sido utilizados como plataforma de demostración en actividades de difusión tecnológica.

Los agentes ESP32 están diseñados como robots de tracción diferencial (DDMR), compuestos por un chasis modular de cuatro piezas (estructura delantera, trasera, inferior y superior). Este diseño facilita la integración de sensores y componentes electrónicos, asegurando estabilidad y precisión en los experimentos. La electrónica de los agentes está montada en una placa de circuito impreso (PCB) diseñada específicamente para alojar los componentes de control y comunicación, optimizando la distribución del hardware y mejorando la confiabilidad del sistema. La Figura 2.1 muestra una vista de las principales componentes mecánicas y electrónicas del agente. Se desplazan mediante dos ruedas activas de 43 mm de diámetro, controladas de forma independiente por motores DC N20, cada uno equipado con un encoder embebido y una caja de reducción con una relación de 100:1. Una bola rodante en la parte frontal inferior estabiliza la estructura del agente.

Para el control longitudinal, se utiliza un sensor frontal de distancia VL53L0X con un campo de visión (FOV) de 25°, que permite medir la distancia a objetos en la línea de vista frontal del agente. Dado que esta medición no es confiable en zonas de curvas, se implementa un algoritmo que ajusta dinámicamente la velocidad de los agentes para evitar colisiones [25]. Este algoritmo limita la velocidad cuando la medición frontal es superior a la referencia establecida, evitando así aceleraciones indebidas que puedan comprometer la estabilidad del pelotón.

En cuanto al control lateral, estos agentes emplean dos arreglos de 8 sensores infrarrojos de reflexión QRE113, ubicados horizontalmente en la parte inferior del agente, uno en la parte delantera y otro en la parte trasera. Estos sensores detectan la posición del agente respecto a la línea central de la pista y miden su orientación con respecto a su eje de traslación. La

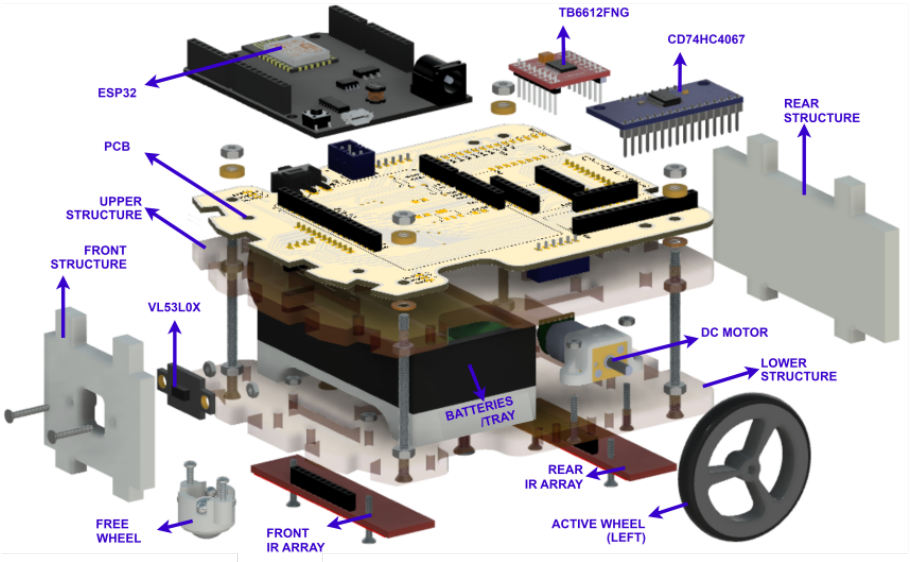


Figura 2.1: Componentes principales de un agente móvil

información de estos sensores se gestiona mediante un multiplexor análogo CD74HC4067, facilitando la adquisición y procesamiento de datos.

Para gestionar la información de estos arreglos de sensores, se emplea un multiplexor análogo CD74HC4067 de 16 canales, facilitando la adquisición y procesamiento de datos, dado que la tarjeta de desarrollo ESP32 no dispone de suficientes canales ADC. La Figura 2.2 muestra el diagrama de interconexión de las principales componentes electrónicas para el accionamiento, sensado y cómputo. Además, los agentes cuentan con encoders Hall-effect integrados en los motores DC, proporcionando mediciones precisas sobre la posición angular, rotación y velocidad angular de cada rueda, lo que permite un control preciso del movimiento y la velocidad del agente.

El agente es energizado por una batería compuesta por dos celdas *Lithium-Ion* 18650 conectadas en serie sin BMS, proporcionando un voltaje nominal de 7.2 V y un máximo de 8.4 V. Cabe señalar que los sensores se alimentan con un voltaje de 3.3 V debido a restricciones de potencia del microcontrolador. La falta de BMS y la incertidumbre sobre la capacidad real de las celdas afectan la autonomía de cada agente, lo que puede generar variabilidad en los experimentos.

Gracias a su conectividad inalámbrica integrada en el microcontrolador ESP32, los agentes pueden comunicarse entre sí para operar de manera autónoma y sincronizada en la formación de un pelotón. Además, interactúan con el monitor/operador mediante una interfaz de usuario dedicada, permitiendo la configuración remota de experimentos, así como el monitoreo y almacenamiento en tiempo real de los datos obtenidos. Estos datos incluyen la etiqueta del agente, que permite identificar y filtrar correctamente la información recibida, la velocidad medida y su referencia, la distancia medida y su valor de referencia, la latencia del ciclo de control y otros parámetros relacionados con el sistema de control, aunque no necesariamente relevantes para su análisis o evaluación.

Los agentes ESP32 utilizan el protocolo UDP para la comunicación inalámbrica, asegurando compatibilidad con la GUI desarrollada, lo que facilita la gestión de parámetros y la visualización

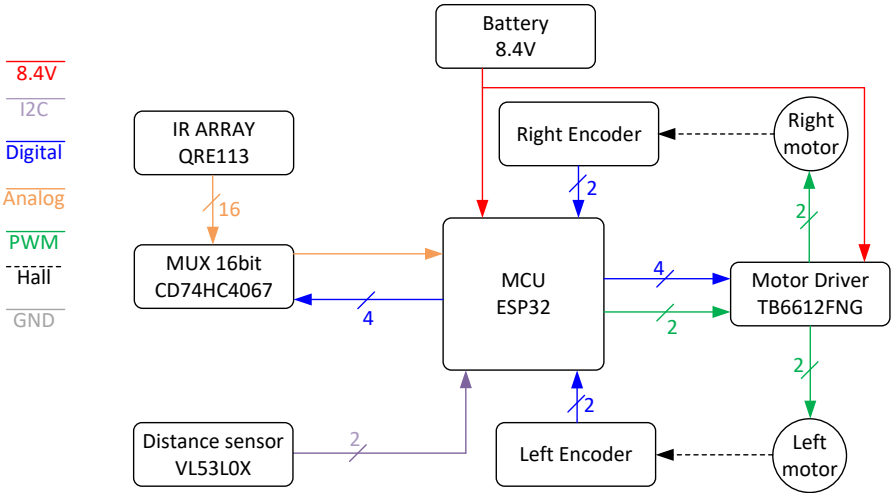


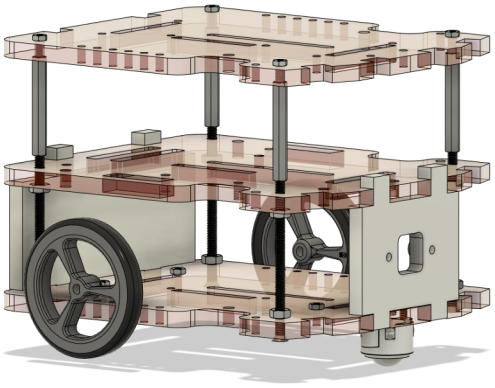
Figura 2.2: Diagrama de bloques de conexión para agentes ESP32

Tabla 2.1: Comandos definidos para el operador de la plataforma experimental.

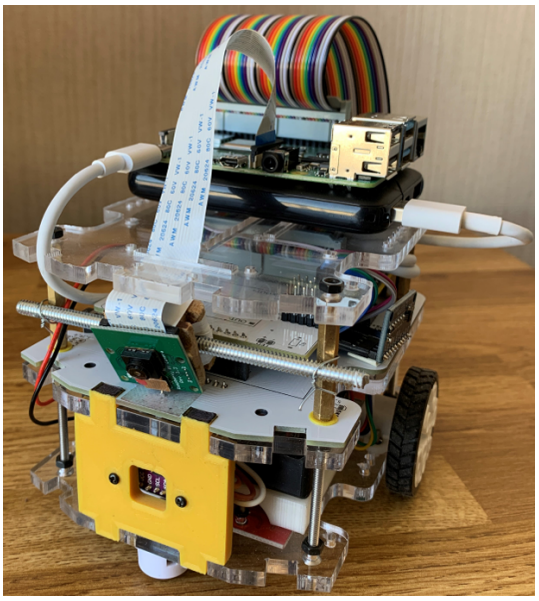
Comando	**Descripción**
E/co_p/valor	Define la ganancia proporcional del controlador de orientación
E/co_i/valor	Define la ganancia integral del controlador de orientación
E/co_d/valor	Define la ganancia derivativa del controlador de orientación
E/cv_p/valor	Define la ganancia proporcional del controlador de velocidad
E/cv_i/valor	Define la ganancia integral del controlador de velocidad
E/cv_d/valor	Define la ganancia derivativa del controlador de velocidad
E/cv_ref/valor	Define la referencia de velocidad
E/cd_p/valor	Define la ganancia proporcional del controlador de distancia
E/cd_i/valor	Define la ganancia integral del controlador de distancia
E/cd_d/valor	Define la ganancia derivativa del controlador de distancia
E/cd_ref/valor	Define la referencia de distancia
E/calibrar/valor	Permite calibrar los arreglos de sensores IR si valor=1
E/parar/valor	Permite iniciar el ciclo de control (`valor=si`) o detenerlo (`valor=no`)

de datos en tiempo real. La estructura de los mensajes UDP válidos y decodificables por los agentes se encuentra detallada en la expresión 2.1, donde E y variable son parámetros específicos documentados previamente en [1]. En dicha documentación, se presenta la Tabla 2.1 con la lista de comandos habilitados e implementados en la interfaz gráfica, en la que el parámetro valor es el único que queda a determinación del usuario, permitiendo flexibilidad en la configuración y control del sistema.

E/variable/valor (2.1)



(a) Estructura versión Raspberry Pi



(b) Implementación completa de Raspberry Pi RUPU

Figura 2.3: versión Raspberry Pi de los agentes

Version Raspberry Pi

Los agentes móviles basados en Raspberry Pi fueron desarrollados como una prueba de concepto para ampliar las capacidades de la plataforma RUPU. A diferencia de la versión ESP32, estos agentes emplean una *Small Board Computer* (SBC) en lugar de un microcontrolador, lo que les permite ejecutar algoritmos más complejos y con mayores requerimientos de procesamiento. Además, la Raspberry Pi cuenta con un puerto de conexión *Flexible Printed Circuits* (FPC) que permite conectar cámaras mediante el protocolo *Camera Serial Interface* (CSI), lo que posibilita la integración de visión por computador para ampliar las capacidades del sistema en tareas de percepción del entorno. Sin embargo, su diseño aún se encuentra en una etapa de prototipo, utilizando como base la estructura de un agente ESP32, como se muestra en la Figura 2.3a. Esto implica desafíos en términos de optimización estructural y compatibilidad con los componentes existentes. Como se observa en la Figura 2.3b, algunos componentes no están fijados a la estructura, las conexiones son visibles y poco robustas, y la distribución de los elementos es deficiente, lo que limita la integración de mejoras de hardware.

La primera versión de los agentes basados en Raspberry Pi comparten muchas características mecánicas con la versión basada en ESP32, utilizando el mismo chasis modular (incluyendo la PCB), sistema de tracción diferencial y sensores, lo que facilita la compatibilidad en términos de hardware. A diferencia de la versión ESP32, estos agentes implementan la comunicación inalámbrica utilizando el protocolo MQTT con el broker *Ubidots*, lo que implica un esquema de comunicación distinto y desafíos adicionales en términos de integración con el resto de la plataforma. La Figura 2.4 muestra el diagrama de bloques de conexión de los componentes, evidenciando que, aunque los sensores son los mismos que en la versión ESP32, se requiere un circuito ADC externo ADS115, ya que la Raspberry Pi no dispone de entradas analógicas nativas. Esta conversión es fundamental para la correcta interpretación de las señales de los

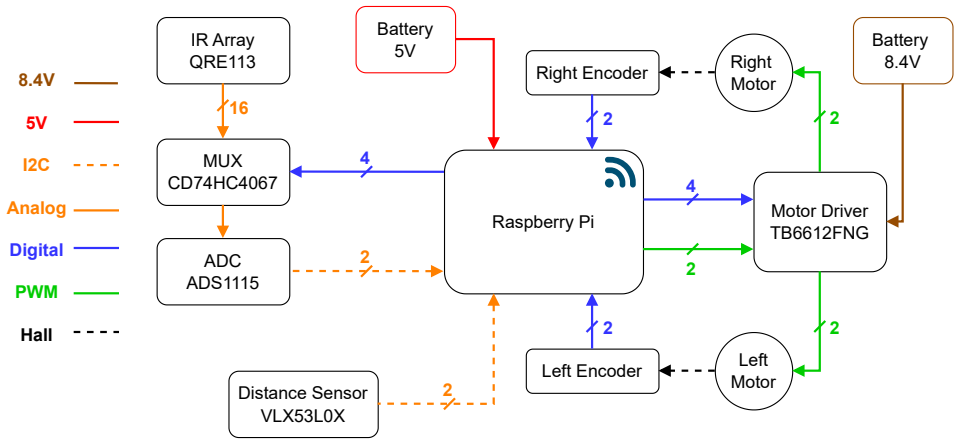


Figura 2.4: Diagrama de bloques de conexión para agentes Raspberry Pi

sensores. Además, esta versión utiliza una fuente de poder independiente exclusivamente para la Raspberry Pi, lo que introduce retos adicionales en la gestión energética y la integración con el resto del sistema, ya que debe cumplir con los requisitos de potencia específicos de la Raspberry Pi para garantizar su funcionamiento estable.

A diferencia de la versión implementada sobre ESP32, la versión basada en Raspberry Pi no ha sido aún validada en escenarios experimentales de control de pelotones ni integrada formalmente con la interfaz gráfica (GUI) desarrollada, lo que introduce desafíos relevantes en términos de integración funcional y verificación del sistema. Este trabajo aborda dichos desafíos, analizando los aspectos críticos que deben ser resueltos para robustecer la implementación sobre Raspberry Pi y cumplir con los requisitos de repetibilidad, estabilidad y escalabilidad definidos para la plataforma.

Algoritmo de control

Ambas versiones de los agentes, ESP32 con Arduino y Raspberry Pi con Python, implementan el mismo algoritmo de control PID. Este esquema de control, cuyo diseño se encuentra documentado en [1], se fundamenta en controladores PID aplicados a las variables de velocidad, orientación y distancia del DDMR, organizados según la notación C1, C2 y C3 en la Figura 2.5. Cada controlador permite la configuración de sus tres parámetros: proporcional, integral y derivativo, los cuales pueden ajustarse directamente en el código o de manera inalámbrica a través de la interfaz de usuario, facilitando la calibración y optimización del sistema en tiempo real sin necesidad de reinicio o recompilación del código. Para el cálculo de la derivada e integral, se emplean aproximaciones numéricas mediante diferencias finitas y la suma trapezoidal, respectivamente. El control opera de forma discretizada con un tiempo de muestreo de 40 ms, definido considerando que el tiempo mínimo de muestreo de los sensores es de 33 ms, proporcionando un margen adicional para mejorar la estabilidad y robustez del sistema [1].

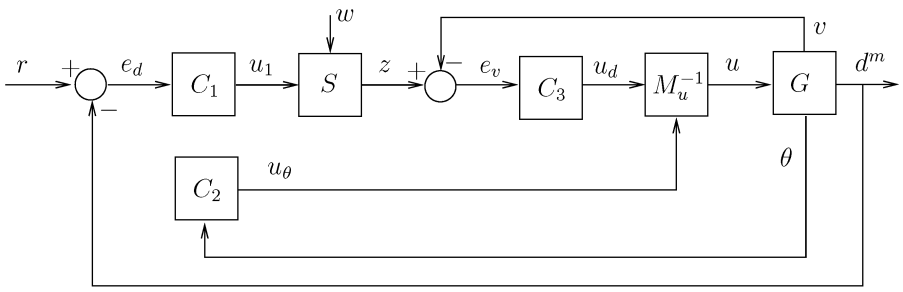


Figura 2.5: Esquema de control implementado en RUPU

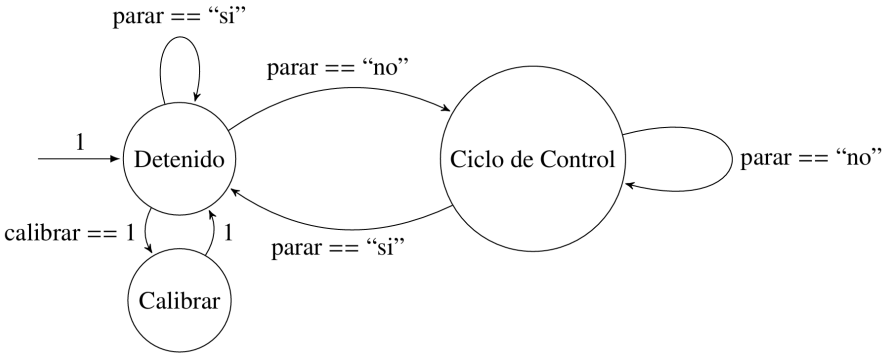


Figura 2.6: Diagrama de estados de operación de los agentes

El esquema de control mostrado se ejecuta a través de un algoritmo, el cual se implementa mediante una máquina de estados finitos (FSM), representada en el diagrama de la Figura 2.6. Cada estado define una fase específica del control, mientras que las transiciones entre ellos dependen tanto de las condiciones operativas del sistema como de las instrucciones del operador. El esquema de control se ejecuta particularmente en el estado “ciclo de control”, detallado en el Listado 2.1. El código consta de una etapa de adquisición de datos de los diferentes sensores. Luego, cada 40 ms, estos datos se procesan para calcular la actuación mediante el control PID, y finalmente se aplican a los motores.

```
def control_loop():
    medir_distancia() # Retorna la medicion en cm del sensor de distancia
    medir_velocidad() # Retorna la velocidad lineal en cm/s
    medir_orientacion() # Retorna el angulo del agente respecto de la linea central en radianes

    tiempo = t_actual - t_controlador
    if tiempo >= 0.04: # Tiempo de muestreo del control
        control_PID() # Calcula la actuacion en base a las mediciones de sensores
        t_controlador = t_actual
    actuacion_motores() # Aplica la actuacion calculada a los motores
```

Listing 2.1: Bucle de control en Python

2.2. Validación experimental de la plataforma

En esta sección se presentan los experimentos realizados para evaluar el desempeño de la plataforma, así como los resultados obtenidos. Se describe la metodología empleada, los criterios de validación y las métricas utilizadas para medir el rendimiento del sistema. En particular, el desempeño se evalúa mediante el análisis de la distancia relativa entre vehículos durante la operación en pelotón, lo que permite cuantificar el comportamiento del sistema en términos de control longitudinal. Por su parte, el control lateral se considera de forma cualitativa, ya que su impacto se manifiesta principalmente en la estabilidad del seguimiento de carril de los agentes móviles.

Para garantizar la confiabilidad de los experimentos realizados en la plataforma, es fundamental evaluar su repetibilidad. Esto implica verificar que, bajo las mismas condiciones de prueba, los resultados obtenidos sean consistentes y comparables entre diferentes iteraciones.

La repetibilidad se evalúa comparando la evolución temporal de la distancia entre vehículos en múltiples repeticiones de un mismo experimento. La presencia de variaciones significativas en esta variable puede evidenciar la influencia de factores externos o limitaciones inherentes al sistema, lo que comprometería la estabilidad de los resultados y, por tanto, la validez de los experimentos.

El análisis de la repetibilidad permite identificar fuentes de error, ya sean sistemáticas o aleatorias, que influyan en el desempeño del sistema. Esto permite no solo validar la robustez de la plataforma, sino también establecer criterios para futuras mejoras en la precisión y fiabilidad de los datos obtenidos.

En el trabajo presentado en [2], se establecieron condiciones de prueba controladas que sugieren una posible sensibilidad de los sensores de los agentes a las variaciones de luminosidad del entorno. A partir de estas observaciones, en este trabajo se propone analizar de forma sistemática dicha sensibilidad como una posible fuente de error que afecta el desempeño del sistema y, en consecuencia, compromete la repetibilidad de los experimentos. Para ello, se diseñaron y ejecutaron una serie de experimentos orientados a caracterizar el impacto de las condiciones lumínicas sobre el comportamiento de los agentes, proporcionando una base para la implementación de mejoras en la plataforma.

Los experimentos fueron diseñados para evaluar el desempeño del sensor de distancia frontal VL53L0X y reevaluar la repetibilidad global de la plataforma. Dado que el funcionamiento de los sensores de los agentes se basa en la reflexión de luz infrarroja, se consideró la iluminación del entorno como una variable crítica en la campaña de medición, permitiendo analizar su impacto en la precisión y estabilidad de las mediciones. A continuación, se detalla la metodología aplicada en cada experimento.

2.2.1. Experimento para evaluar desempeño de sensor de distancia VL53L0X

El experimento se realiza en un laboratorio equipado con persianas que permiten el bloqueo parcial de la luz natural externa, lo que facilita el control de las condiciones lumínicas del entorno. Además, se utiliza una linterna como fuente de luz artificial focalizada para introducir perturbaciones específicas durante la prueba. Durante la realización del experimento, se planean distintos cambios en la iluminación ambiental con el fin de evaluar el desempeño del sensor

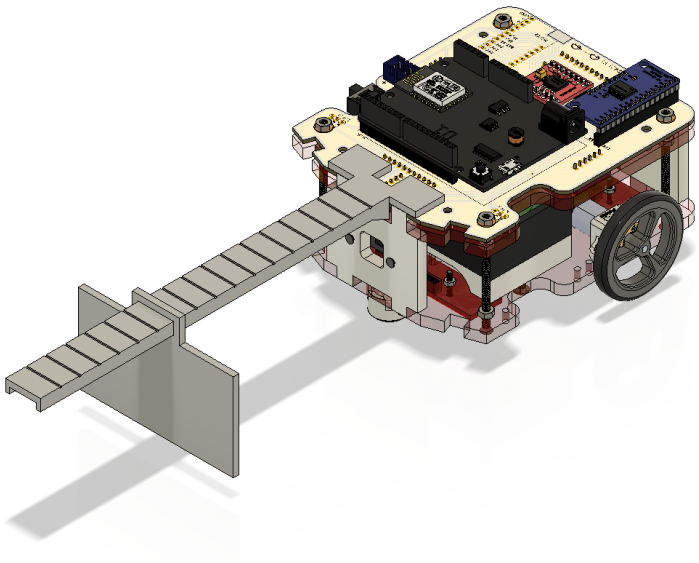


Figura 2.7: Agente modificado para experimento

ante variaciones y perturbaciones en las condiciones de luz. La prueba se lleva a cabo en una pista con forma de óvalo cerrado, construida con baldosas de espuma negra y una línea central blanca de 2 cm de ancho.

Para la evaluación, se emplea un único agente modificado mediante la incorporación de una pieza plástica graduada en centímetros, diseñada para proporcionar una referencia de distancia constante y ajustable entre experimentos, como se muestra en la Figura 2.7. El carro se configura como líder con una velocidad fija de 10 cm/s y una distancia de medición ajustada a 10 cm.

Los datos se registran bajo diferentes condiciones de iluminación, obteniendo el gráfico de la Figura 2.8. En este, se destacan cuatro secciones, que corresponden a los cambios efectuados en la luz ambiental durante el experimento. Tras las condiciones iniciales ya descritas, se abren las persianas, permitiendo que la luz natural incida directamente sobre el agente y la pista. Luego, estas se cierran y se introduce una fuente de luz artificial mediante un LED blanco, el cual se dirige de manera constante y directa al sensor de distancia durante todo el recorrido de la pista. Posteriormente, el LED se activa de forma intermitente en distintos tramos, y finalmente, se repite este procedimiento con las persianas abiertas.

A partir del gráfico, se puede apreciar un desfase vertical de aproximadamente 1 cm, lo que se debe a la ubicación del sensor, el cual está montado a esta distancia por detrás del extremo frontal del robot. Esta diferencia en la medición es esperada y no representa una fuente de error en la repetibilidad del sistema, sino una característica propia de la disposición del sensor en la estructura del agente y debe tenerse en cuenta en el procesamiento posterior de las mediciones.

Además, se observa la presencia de ruido significativo en la señal, lo cual puede atribuirse a la tolerancia ($\pm 5\,\%$) y precisión del sensor [26] y a la duración del experimento, ya que pequeños desajustes en la lectura son normales en sensores de esta naturaleza. A pesar de esto, el comportamiento de la medición se mantiene consistente, sin variaciones significativas atribuibles a cambios en las condiciones lumínicas del entorno. Esto sugiere que la iluminación

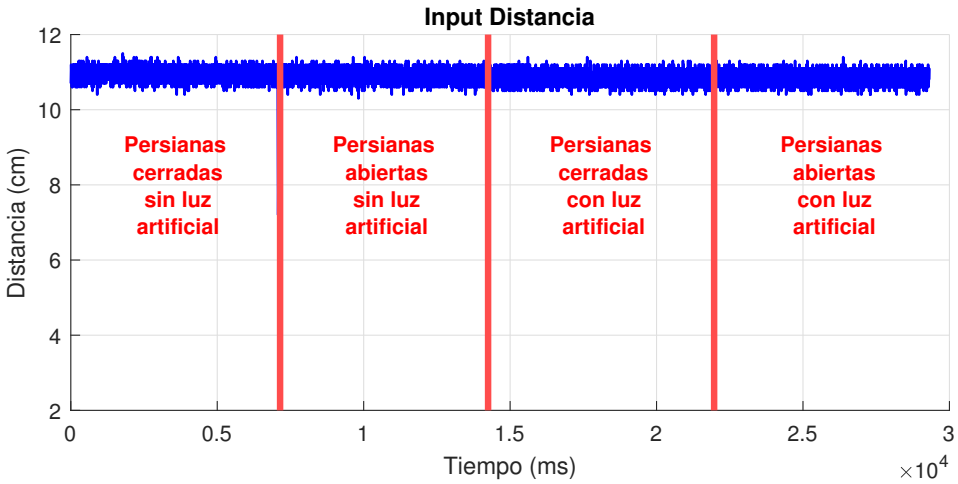


Figura 2.8: Mediciones de distancia registradas en experimento bajo diferentes condiciones ambientales

no es una fuente de error que afecte la repetibilidad de la plataforma.

2.2.2. Experimento para evaluar repetibilidad - arreglo QTR

El experimento para evaluar la repetibilidad global de la plataforma se llevó a cabo en un laboratorio bajo las mismas condiciones lumínicas iniciales que el experimento anterior. Para ello, se utilizaron cuatro agentes basados en ESP32, configurados para operar en formación de pelotón con una separación inicial de 10 cm, una distancia de referencia fija de 10 cm y una velocidad de referencia fija de 12 cm/s.

La campaña de medición se realizó en una pista cerrada con secciones rectas y curvas de distintos largos, construida con las mismas baldosas de espuma mencionadas anteriormente, como se muestra en la Figura 2.9. Durante el experimento, se registró la distancia medida por cada agente mientras completaban 9 vueltas a la pista, repitiendo la prueba en 6 iteraciones para evaluar la consistencia de los resultados.

Las mediciones de distancia típicas registradas por un agente seguidor al completar una vuelta en la pista se muestran en la Figura 2.10, en este gráfico se observa que la medición presenta variaciones significativas en ciertas zonas. Estas variaciones corresponden a momentos en los que el agente pierde la visión del vehículo que le precede, lo que implica que el control longitudinal no depende de la medición de este sensor, por lo que las mediciones registradas en estas regiones no deben considerarse para evaluar el desempeño de los agentes.

Así, se identifican dos casos de interés para evaluar el desempeño de la plataforma. Cuando los agentes se encuentran en línea recta, el control longitudinal se basa en la medición de distancia, mientras que cuando los agentes se encuentran en una curva, este control utiliza como referencia la velocidad del agente predecesor complementado con un algoritmo que limita la velocidad máxima para evitar colisiones presentado en [25].

En la Figura 2.11, se destacan los diferentes segmentos de la pista y su relación con la distancia medida por el agente. Para obtener métricas representativas del desempeño, es necesario

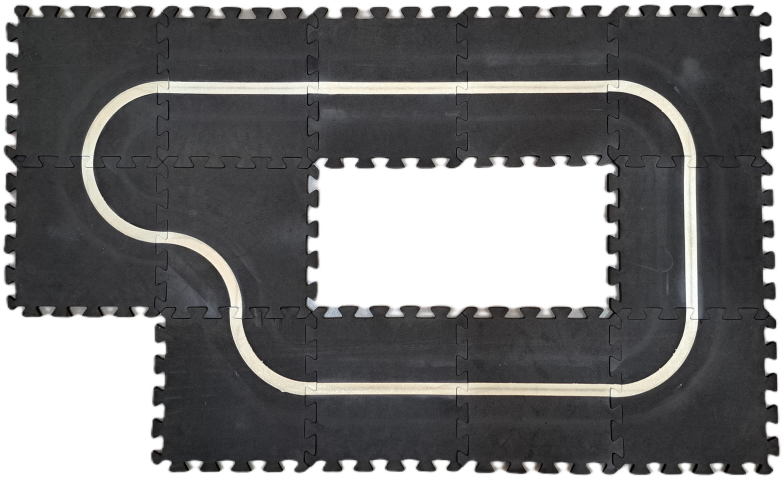


Figura 2.9: Pista utilizada para experimentos de repetibilidad

filtrar las secciones de color verde, rojo y amarillo, ya que en estas zonas el agente pierde la referencia de su predecesor, por lo tanto, solo deben considerarse las secciones azules, donde el control longitudinal opera basado en la medición de distancia, permitiendo una evaluación más rigurosa del desempeño del sistema.

Para mejorar la calidad de las mediciones, reducir la influencia de ruidos e interferencias en los datos y poder realizar un análisis adecuado de la repetibilidad de la plataforma, se implementó un proceso de filtrado de señales utilizando MATLAB. Este proceso incluye la eliminación de valores elevados que podrían afectar el análisis, la reducción de variabilidad mediante varianza móvil y la alineación de señales utilizando correlación cruzada y Dynamic Time Warping (DTW). Los scripts utilizados para aplicar este procesamiento se encuentran en el repositorio GitHub que complementa el trabajo realizado en este trabajo de memoria .

Inicialmente, se cargaron los datos obtenidos de diferentes sensores y se estableció un umbral límite para eliminar valores que superan los rangos esperados, evitando así posibles interferencias en el proceso de filtrado. Posteriormente, se aplicó un análisis de varianza móvil para identificar regiones con alta variabilidad, es decir, regiones donde las mediciones no son confiables para evaluar la repetibilidad, eliminando aquellos segmentos cuya varianza superaba un umbral predefinido. El umbral predefinido utilizado en este análisis fue determinado a partir del valor máximo de varianza registrado en las regiones donde las mediciones deben ser consideradas para el análisis, permitiendo así una segmentación más precisa de los datos relevantes. Finalmente, se empleó la técnica de alineación de señales *Dynamic Time Warping* (DTW), ya que la tendencia de las señales es similar, pero existe un desfase temporal producido por la pérdida de datos inherente al canal de comunicación y por retrasos atribuibles al operador. Esto puede explicarse porque cada iteración no inicia al mismo tiempo desde que se capturan los datos y porque los agentes no inician ni detienen la ejecución en los mismos puntos. No obstante, estos retrasos y pérdidas de datos no deben ser considerados en el análisis de repetibilidad, ya

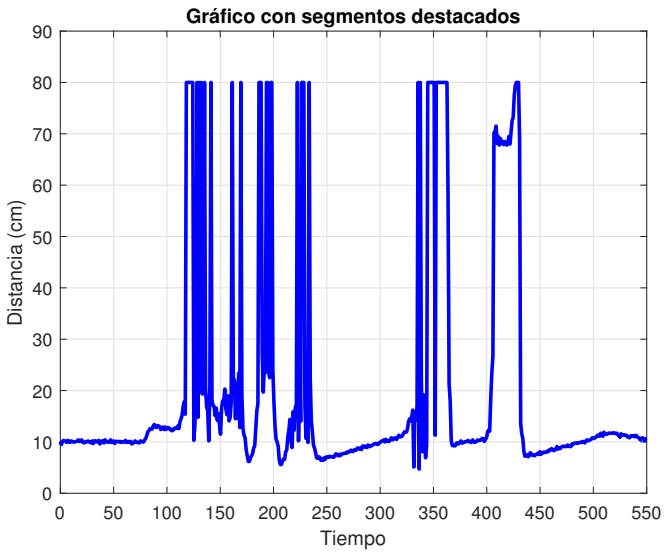


Figura 2.10: Medición de distancia en una vuelta de la pista utilizada

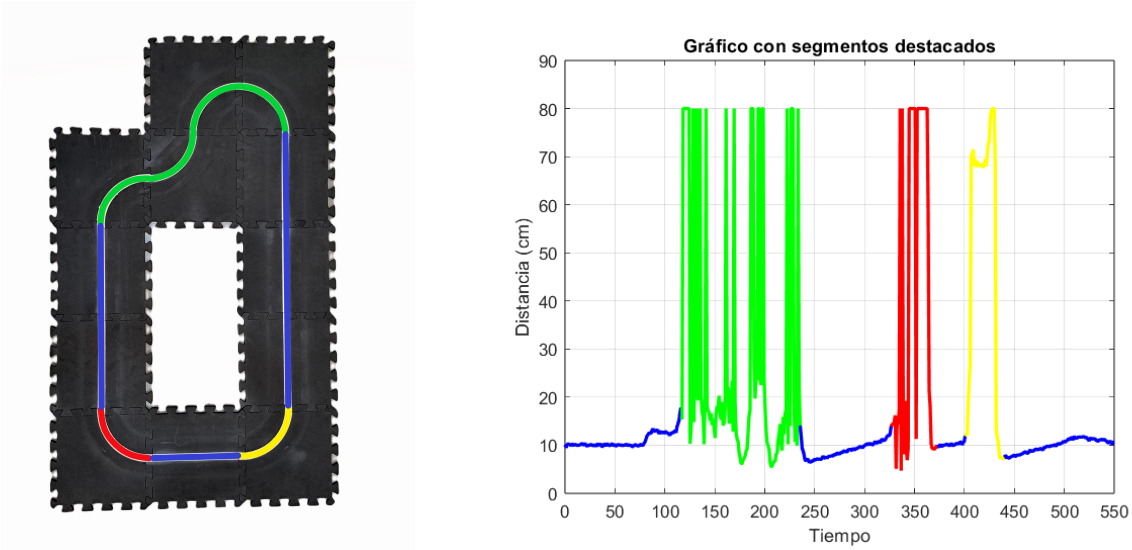
que corresponden a factores externos que no reflejan el desempeño intrínseco de la plataforma. Los resultados obtenidos tras la aplicación del procesamiento pueden observarse en la Figura 2.12.

Visualmente, no se observan diferencias apreciables en los gráficos procesados, lo que indica que la alineación y el filtrado de señales han sido efectivos. Se distinguen claramente las secciones señaladas en la figura 2.11b, así como las distintas vueltas realizadas por los agentes. A partir de estos resultados, se empleará la raíz del error cuadrático medio normalizado (RMSE normalizado), definida en la ecuación 2.2, como métrica para evaluar la repetibilidad del sistema. Esta medida cuantifica la variabilidad de las mediciones respecto a una referencia establecida, y se normaliza utilizando el rango de la señal de referencia, lo que permite expresar el error en términos porcentuales. Esto facilita la interpretación y permite comparar el nivel de desviación relativa entre distintas repeticiones.

$$\mathrm{RMSE}_{\mathrm{norm}} = \frac{\sqrt{\frac{1}{n}\sum i = 1^{n}(y_i - \hat{y}_i)^2}}{\mathrm{máx}(\hat{y}) - \mathrm{mín}(\hat{y})} \times 100 \tag{2.2}$$

El RMSE normalizado se calculará para cada agente y cada iteración experimental, tomando como referencia la primera repetición del experimento. Esto permitirá cuantificar el nivel relativo de desviación de las mediciones y evaluar la estabilidad del sistema bajo condiciones controladas. Los valores obtenidos se presentan en la Tabla 2.2, entregando una caracterización cuantitativa de la repetibilidad de la plataforma.

Los valores de RMSE normalizado presentados en la Tabla 2.2 se mantienen por debajo del 3,1 % para todos los agentes y repeticiones. Mediante una validación visual del comportamiento, se constata un control adecuado tanto en el seguimiento de la línea demarcada como en el mantenimiento de la distancia entre carros, sin oscilaciones perceptibles. Esta evaluación cualitativa permite concluir que el nivel de error RMSE es aceptable para el contexto de la aplicación. Esta consistencia refleja un comportamiento estable del sistema en condiciones experimentales



(a) Pista segmentada según los casos de interés

(b) Medición con tramos destacados según su relación con el segmento de la pista

Figura 2.11: Segmentación de la pista y su relación con la distancia medida

Tabla 2.2: Resultados de RMSE según iteraciones para diferentes carros

Carro \ iteración N°	2	3	4	5	6
2do carro	1.8809 %	2.0401 %	2.0306 %	2.0213 %	1.9288 %
3er carro	2.7807 %	2.8423 %	2.7950 %	3.0970 %	3.0970 %
4to carro	2.1110 %	1.9970 %	1.9447 %	1.8430 %	1.7490 %

controladas, por lo que las diferencias observadas pueden considerarse despreciables al evaluar el desempeño general de la plataforma.

Los experimentos realizados permiten concluir que la plataforma actual es capaz de producir resultados repetibles en entornos controlados, incluso bajo condiciones menos estrictas que las planteadas en [2]. Esto sugiere que la infraestructura base disponible, en particular los sensores actualmente integrados en los agentes móviles, no requiere modificaciones a nivel de hardware para garantizar la repetibilidad de los experimentos en las condiciones evaluadas. En consecuencia, no se considera necesario reemplazar los sensores en el desarrollo de la nueva versión de los agentes móviles, lo que permite enfocar los esfuerzos de mejora en otros aspectos de la plataforma.

2.3. Mejoras propuestas

A partir de las evaluaciones desarrolladas y el estudio detallado de los componentes de la plataforma junto a un análisis de las necesidades de la versión actual, se proponen mejoras con el requisito fundamental de que ambas versiones de los agentes sean funcionalmente compatibles, especialmente en lo referente al desempeño dinámico individual y a los protocolos

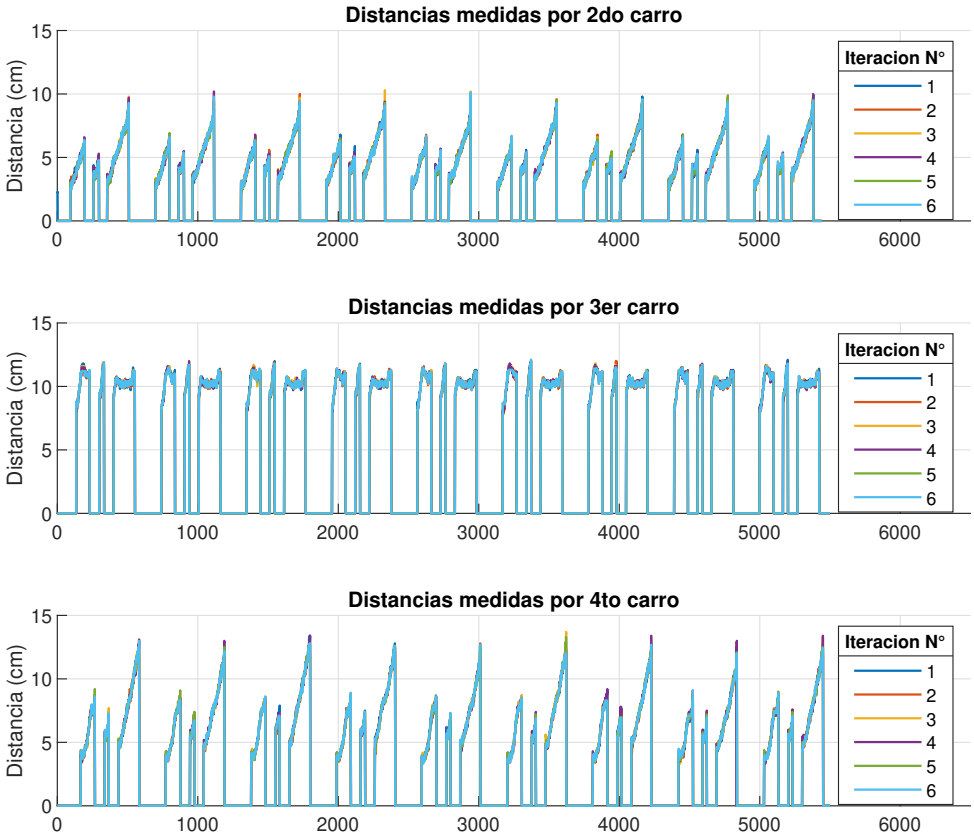


Figura 2.12: Resultados experimento repetibilidad

de comunicación utilizados entre agentes.

Estas propuestas están orientadas a mejorar tanto los aspectos técnicos como operativos, garantizando una mayor funcionalidad y usabilidad de la plataforma. Asimismo, se asegura que las mejoras propuestas no limiten la implementación de futuros desarrollos. Las propuestas se centran en los siguientes aspectos:

- **Protocolo de comunicación**: Como se mencionó anteriormente, los agentes basados en Raspberry Pi, desarrollados en [3], implementa la comunicación inalámbrica mediante el protocolo MQTT, configurado para operar con la plataforma Ubidots. Sin embargo, la falta de documentación asociada y de credenciales de acceso imposibilita su integración y operatividad dentro del sistema. Por esta razón, resulta fundamental modificar el protocolo de comunicación para garantizar la compatibilidad con la interfaz de usuario de la plataforma RUPU, desarrollada en [2]. Esta modificación no solo mejoraría la usabilidad de los agentes basados en Raspberry Pi y facilitaría la ejecución de experimentos, sino que también permitiría, como objetivo final, la implementación de un pelotón de agentes heterogéneos, independiente de la unidad de cómputo utilizada en cada agente.
- **Diseño de la estructura**: El rediseño de la estructura de los agentes basados en Raspberry Pi es necesario para mejorar la estabilidad mecánica y optimizar la distribución de los componentes electrónicos. Actualmente, el uso de la estructura original de los agentes ESP32 genera limitaciones en la disposición de los elementos, dificultando la

fijación adecuada de los componentes y la protección de las conexiones eléctricas. Esto no solo compromete la roño optimizado permitiría mejorar la organización del cableado, garantizar la correcta integración de la Raspberry Pi en la plataforma y facilitar la implementación del resto de las mejoras propuestas o aquellas que puedan desarrollarse en trabajos futuros.

- **Fuente de Poder**: Actualmente, los agentes de la plataforma RUPU utilizan como fuente de alimentación dos celdas de batería conectadas en serie y sin un sistema de gestión de baterías (BMS). La incorporación de un BMS es fundamental para mejorar la seguridad, eficiencia y durabilidad de las celdas [27, 28], ya que permite un control adecuado del voltaje y la corriente, evitando sobrecargas y descargas profundas que puedan comprometer el rendimiento del sistema. Implementar esta solución optimizará el rendimiento general de los agentes, garantizando una fuente de alimentación estable y confiable.

  Para la correcta integración de este sistema, es esencial realizar una medición de la capacidad de cada celda de batería, asegurando que operen en pares con capacidades similares. Esto permitirá mejorar la eficiencia energética y prolongar la vida útil de las celdas, asegurando un desempeño uniforme entre los agentes.

  Por otra parte, como se muestra en la Figura 2.4 los agentes basados en Raspberry Pi cuentan con una batería independiente destinada exclusivamente a la alimentación de esta placa. Sin embargo, esta configuración resulta ineficiente, tanto en términos energéticos como de espacio, además de afectar la usabilidad general de la plataforma. Por ello, es necesario incorporar una única fuente de alimentación que cumpla con los requisitos energéticos de la Raspberry Pi y permita su integración en la plataforma sin mayores dificultades y/o modificaciones.

- **Topología de Red**: La topología de red utilizada en la plataforma RUPU incluye un componente externo a los agentes y al monitor, el cual consiste en un router encargado de configurar y administrar la red. Sin embargo, este dispositivo requiere una fuente de alimentación propia y su configuración varía según el fabricante y modelo, lo que impide generar documentación estandarizada para su implementación. Como consecuencia, el uso de este componente se convierte en un factor crítico que afecta la replicabilidad de la plataforma.

  Para mejorar la replicabilidad y usabilidad, se propone configurar la Raspberry Pi de uno de los agentes como punto de acceso, permitiendo que esta funcione como reemplazo del router. Esta solución facilita la documentación y estandarización del proceso de configuración de la red, asegurando que cualquier implementación futura pueda replicarse sin depender de hardware externo específico.

3 | Implementación de mejoras

A partir del análisis realizado en el capítulo anterior, se proponen una serie de mejoras orientadas a ampliar la usabilidad, reproducibilidad y replicabilidad de la plataforma, con un enfoque particular en los agentes basados en Raspberry Pi.

En este capítulo se detalla el proceso de implementación de estas mejoras, abordando los desafíos técnicos asociados y definiendo los criterios de diseño y evaluación que guían su integración en la plataforma.

3.1. Compatibilidad en el protocolo de comunicación

Para comenzar a trabajar en las mejoras propuestas para los agentes basados en Raspberry Pi, se determina que el primer paso es lograr la compatibilidad del protocolo de comunicación con el resto de los componentes de la plataforma, ya que hasta el momento se desconoce el procedimiento para enviar instrucciones a la Raspberry Pi. Esto implica reemplazar la comunicación basada en Ubidots por el protocolo UDP, asegurando una integración uniforme con el sistema.

Inicialmente, se realiza un análisis detallado del script en Python desarrollado en [3], donde se emplea Ubidots junto con una interfaz tipo Dashboard para el envío de instrucciones a los agentes, incluyendo los parámetros de las constantes de los controladores PID y sus respectivos valores de referencia.

A partir de este análisis, se modifica el script original para eliminar toda dependencia del protocolo MQTT y de Ubidots, reemplazándolo por sockets UDP configurados para enviar y recibir mensajes según los puertos definidos en [1]. Además, se implementan las funciones necesarias para reconocer los comandos válidos establecidos en el mismo trabajo y modificar las variables asociadas a estos. Dichos comandos se presentan en la Tabla con comandos 2.1.

Se realizan las primeras pruebas de funcionamiento del agente disponible basado en Raspberry Pi en una pista similar a la ya utilizada en los experimentos de repetibilidad, mostrada en la Figura 2.9 junto a la interfaz de usuario propia de la plataforma. Visualmente se aprecia un mal desempeño del carro en cuanto al control lateral, evidenciando oscilaciones importantes sobre la línea central a bajas velocidades. Esto se respalda con los datos registrados que se presentan en el gráfico de la Figura 3.1.

Como se observa en la Figura 3.1, al aumentar la velocidad del agente el sistema presenta inestabilidad tanto en el control lateral utilizado para el seguimiento de carril como en el de velocidad. En el gráfico que presenta el ángulo de desviación se puede evidenciar este fenómeno,

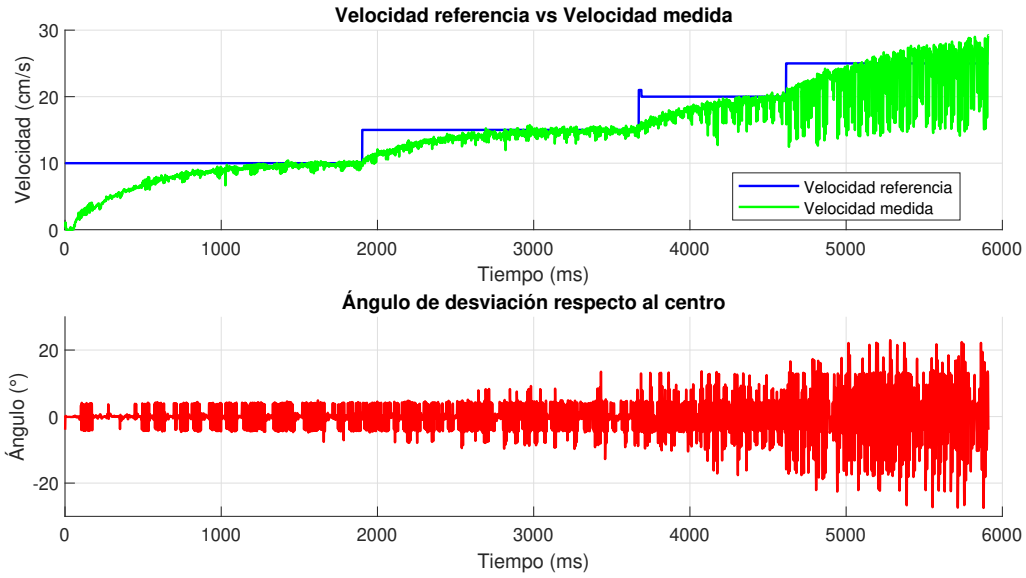


Figura 3.1: Desempeño inicial versión Raspberry Pi

pues los ángulos alcanzan valores superiores a 20°, lo cual es elevado considerando que los agentes sólo permiten una desviación máxima de 25°antes de que el arreglo de sensores inferior pierda la visión de la línea.

Junto a ello, se obtuvo un tiempo de muestreo promedio de los algoritmos de control de aproximadamente 80 ms según los datos registrados por la interfaz de usuario de la plataforma. Este valor es el doble del período de muestreo original (40 ms) definido para el controlador base utilizado en los agentes basados en ESP32 (ver Sección 2.1.1). Este incremento en el tiempo de muestreo afecta el rendimiento del sistema, ya que reduce la frecuencia con la que se actualizan las actuaciones calculadas por el algoritmo de control, lo que puede llevar a respuestas más lentas ante cambios en el entorno y una menor capacidad para corregir errores en tiempo real. Como resultado, se observa una menor estabilidad en el seguimiento de trayectoria y un incremento en las oscilaciones del vehículo. Esto podría explicar el desempeño observado en la Figura 3.1, ya que los parámetros de control fueron sintonizados considerando un tiempo de discretización fijo de 40 ms [1].

3.1.1. Optimización en la adquisición de datos

Con el objetivo de optimizar el script, identificar y analizar cuellos de botella en el desempeño general de los algoritmos e intentar reducir el tiempo de muestreo del control, se emplea la técnica de Profiling de software. Este proceso implica la recolección y análisis de métricas durante la ejecución del programa para detectar de manera determinista las deficiencias en el rendimiento. Estas deficiencias pueden estar relacionadas con uso excesivo de memoria, ineficiencia en el procesamiento y algoritmos mal optimizados, lo que permite realizar ajustes precisos para mejorar la eficiencia del sistema.

Python ofrece diversas herramientas para evaluar el rendimiento del código, lo que permite identificar ineficiencias en los algoritmos y mejorar su desempeño. Entre los profilers más

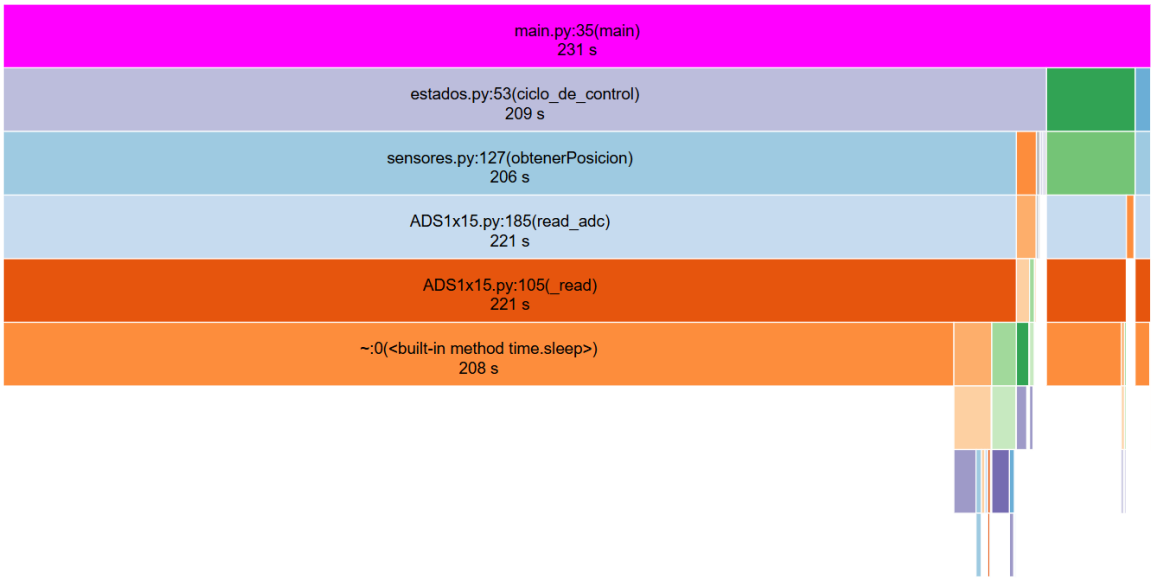


Figura 3.2: Estadísticas de profiling.

utilizados se encuentran **cProfile** [29], **profile** [29] y **timeit** [30]. cProfile es una de las opciones más robustas y ampliamente utilizadas, ya que está implementado en C, lo que reduce el overhead en comparación con otras herramientas y permite obtener información detallada sobre la cantidad de llamadas a funciones y el tiempo de ejecución de cada una. profile, aunque similar, es una alternativa implementada completamente en Python, lo que lo hace más flexible pero introduce un mayor impacto en el rendimiento. Por otro lado, timeit es útil para medir el tiempo de ejecución de pequeñas secciones de código, pero no proporciona una visión completa del rendimiento general del programa. En este trabajo se optó por utilizar **cProfile**, ya que, al operar a un nivel más bajo, requiere menos intervención directa en el código y, por lo tanto, interfiere en menor medida con las mediciones de rendimiento.

Sin embargo, el uso de profilers presenta ciertas limitaciones. Una de las principales es la sobrecarga en el tiempo de ejecución, ya que la inserción de mediciones en el código puede incrementar el uso de recursos, aumentar la latencia del programa y afectar la precisión de los resultados.

En la Figura 3.2, se muestran los datos recolectados al utilizar el profiler **cProfile** en el script disponible. Estos se presentan de forma jerárquica en un gráfico tipo *Icicle*, donde las funciones de mayor nivel aparecen en la parte superior, mientras que las funciones llamadas por estas se representan en los niveles inferiores. El ancho de cada barra en el gráfico indica la cantidad de tiempo consumido por cada función, permitiendo identificar los procesos más demandantes dentro del código.

Este gráfico presenta la función *main.py* en el nivel superior con un tiempo de 231 segundos, esto indica que el script ejecutado corresponde a *main.py* y el experimento tuvo una duración total de 231 segundos. Se identifica de inmediato que el llamado a la función *time.sleep()* representa un cuello de botella en el rendimiento [1], ya que consume aproximadamente el 90 %

[1]Cabe señalar que la función *time.sleep()* se considera una función de temporización en sistemas en tiempo real. Sin embargo, dado que la Raspberry Pi opera con una distribución de Linux, que no es un sistema operativo

del tiempo total de ejecución del programa. Esta función está en un nivel jerárquico inferior a la función *read_adc()*, es decir, es ejecutada en esta función, la cual se encarga de leer los valores de los canales del convertidor analógico-digital ADS115 utilizado en el sistema. Al realizar un análisis detallado sobre el funcionamiento de este circuito, se identificó que la biblioteca utilizada para este circuito [31] define el tiempo de muestreo mediante la ecuación 3.1.

$$\texttt{time.sleep()} = \frac{1}{\texttt{data\_rate}} + 0{,}0001 \tag{3.1}$$

En esta ecuación, *data_rate* representa la tasa de muestreo del sensor y puede configurarse en valores discretos entre 128 (por defecto) y 860 SPS (samples per second). Sin embargo, tras realizar pruebas en condiciones de operación estándar, incluyendo la lectura continua de datos y la sincronización con el arreglo de sensores y multiplexor, se obtuvo un máximo de 300 SPS de mediciones válidas sin redundancias, lo que indica una reducción significativa respecto al valor teórico.

Debido a esta limitación, se justifica la búsqueda de una alternativa más adecuada de este circuito para la aplicación de seguimiento de carril, que proporcione una tasa de muestreo efectiva mayor que la actual. Para ello, se toman como referencia las características del microcontrolador ESP32, el cual cuenta con un ADC de 12 bits y una tasa de datos de 9600 SPS. Estas especificaciones sirven como criterio de selección para evaluar un reemplazo para el convertidor analógico-digital usado, entre las opciones consideradas se encuentran:

- ADS1115 [32]: ADC de 16 bits, cuatro canales de entrada y una tasa de muestreo de hasta 860 SPS. Utiliza una interfaz I²C, que es más escalable y permite la conexión de múltiples circuitos que implementen la misma interfaz, esto facilita su integración con la Raspberry Pi. Sin embargo, la velocidad de adquisición está limitada por el protocolo I²C, lo que puede afectar aplicaciones que requieren tiempos de muestreo más rápidos. Este ADC también se basa en una arquitectura Sigma-Delta, lo que le permite obtener mediciones más precisas al filtrar ruido, aunque a costa de una menor velocidad de conversión.
- ADS1015 [33]: Variante del ADS1115 con una resolución de 12 bits y una mayor tasa de muestreo de 3.3 kSPS. Comparte la cantidad de canales de entrada, la interfaz I²C y la arquitectura Sigma-Delta, ofreciendo buena precisión pero con una menor capacidad de resolución en comparación con el ADS1115. Su ventaja principal es la mayor velocidad de conversión, mientras que su desventaja sigue siendo la limitación en la velocidad de adquisición impuesta por I²C.
- MCP3208 [34]: ADC de 12 bits con ocho canales de entrada y una tasa de muestreo de 100 kSPS. Utiliza una interfaz SPI, lo que permite una adquisición de datos más rápida y eficiente en Raspberry Pi. A diferencia de los ADS1015 y ADS1115, este ADC emplea una arquitectura SAR (Successive Approximation Register), lo que permite una conversión mucho más rápida en comparación con los Sigma-Delta. Su principal ventaja es la mayor velocidad de muestreo y la disponibilidad de más canales en comparación con las opciones I²C. Sin embargo, su implementación es más compleja debido a la gestión del protocolo SPI. Su voltaje de operación oscila entre 2.7V y 5.5V, lo que lo

en tiempo real, la precisión de esta función se ve afectada

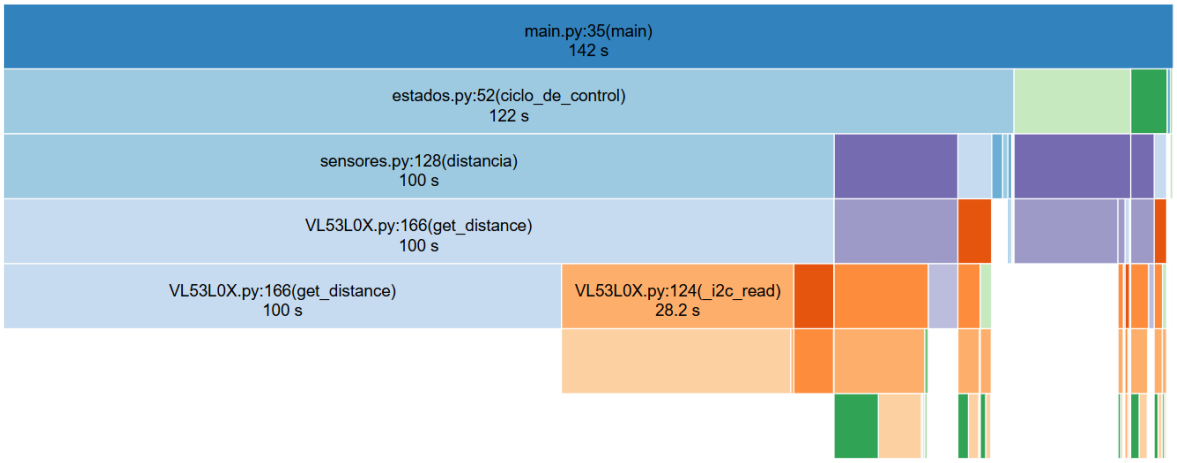


Figura 3.3: Estadísticas de profiling con ADC MCP3208

hace una opción versátil para aplicaciones que requieren múltiples mediciones analógicas con mayor rapidez de conversión.

En aplicaciones de agentes móviles con seguimiento de carril, un ADC con arquitectura SAR como el MCP3208 es generalmente más adecuado, ya que ofrece adquisición de datos con baja latencia, reduciendo el tiempo de conversión en cada ciclo de control y permitiendo destinar el margen sobrante a otros procesos. . Además, al contar con 8 canales, este circuito puede conectarse directamente a cada arreglo de sensores de los agentes, eliminando la necesidad de un multiplexor y simplificando la implementación del sistema.

Para validar el desempeño del MCP3208, se incorporó este ADC en un agente RUPU basado en Raspberry Pi. Tras ensamblar y configurar el sistema, se realizaron pruebas funcionales (como la lectura concurrente del arreglo de sensores y envío de datos) para verificar su correcta integración con los sensores y la transmisión de datos. Posteriormente, se llevaron a cabo experimentos en pista, donde se observó una mejora visual en el desempeño del sistema. Aunque visualmente el desempeño del sistema mejoró, el análisis de los datos enviados por el agente reveló que el tiempo de ejecución del ciclo de control aún se encuentra entre 60 y 70 ms, lo que sigue siendo superior al tiempo objetivo de 40 ms. Debido a esto, se justificó la realización de un nuevo análisis de profiling para seguir optimizando el rendimiento del sistema. La Figura 3.3 muestra los resultados de profiling, en donde se reporta el tiempo total de ejecución del ciclo de control, así como la distribución del tiempo de procesamiento entre las distintas funciones que se ejecutan en este ciclo. Los resultados indican que más del 50 % (corregir el porcentaje) del tiempo de ejecución del lazo de control se emplea en las funciones relacionadas con el sensor de distancia VL53L0X. En la Tabla 3.1 se presentan mas detalles del profiling realizados en estos experimentos. Cabe señalar que se utilizó una función vacía denominada *control_aux()*, la cual se invoca al mismo tiempo que la función del control PID, luego de la línea 7 del código mostrado en el Listado2.1, permitiendo que la herramienta identifique cuando se actualizan las salidas del esquema de control en cascada para así analizar de manera más detallada el comportamiento de los algoritmos de adquisición de datos. Es importante destacar que *control_aux()* al tratarse de una función vacía es poco relevante para el desempeño general de los algoritmos, teniendo un tiempo acumulado de 0.001 ms.

Las estadísticas de la Tabla 3.1 muestran que en cada iteración del ciclo de control se realiza

Tabla 3.1: Estadísticas de profiling relevantes en el ciclo de control

Función	N° llamados	Tiempo medio (s)	Tiempo acumulado (s)
`ciclo_de_control()`	3959	0.03091	122.4
`get_distance()`	3959	0.02536	100.4
`control_aux()`	1980	0	0.001

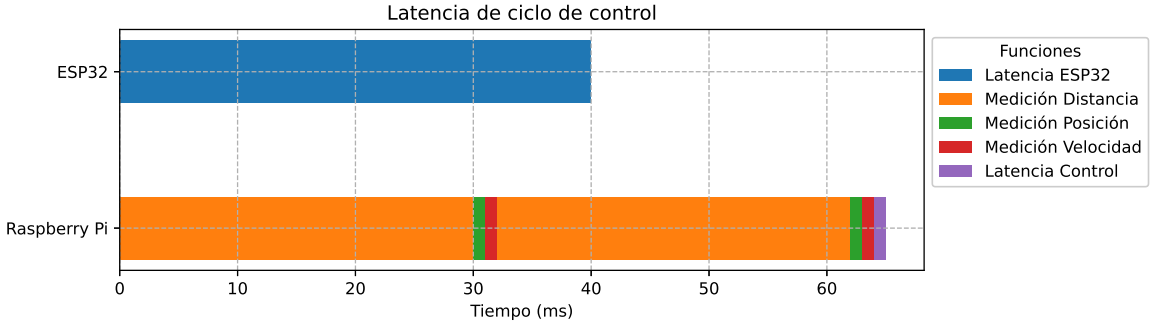


Figura 3.4: Tiempos de latencia ciclo de control

una medición de distancia. En efecto, tanto la función principal del ciclo de control como la función de lectura de distancia (*get_distance()*) se invocan 3959 veces, sin embargo, la función auxiliar que indica que se ejecuta el cálculo de la actuación a través del controlador PID, *control_aux()*, es llamada sólo 1980 veces, es decir, el cálculo del control sólo se realiza la mitad de las veces que se ejecuta todo el ciclo, Esto evidencia que la adquisición de datos de los sensores y la ejecución del PID se realizan a frecuencias distintas.

Por otro lado, la función relacionada a la lectura del sensor de distancia (*get_distance()*) ocupa cerca del 82 % del tiempo de latencia promedio del ciclo (25,36 ms de 30,91 ms), esto sugiere que podría existir un margen para mejorar la sincronización y el uso de los recursos del sistema. Según estos datos y el código mostrado en el Listado 2.1, se realizan dos mediciones de distancia antes de calcular una nueva actuación a través del control PID. La primera medición tarda en promedio 30 ms, lo que no cumple con la condición mostrada en la línea 7, y la segunda medición alcanza una latencia total de aproximadamente 62 ms, lo que cumple con la condición, pero resulta en un tiempo de muestreo mayor al tiempo objetivo. La Figura 3.4 muestra una representación de la latencia completa del ciclo de control, basada en las estadísticas de la Tabla 3.1, lo que ilustra el fenómeno descrito anteriormente.

El sensor de distancia VL53L0X se puede configurar para diferentes tasas de muestreo, existiendo un compromiso entre tiempos de muestreo más cortos y la precisión del sensor. Por ejemplo, en modo `HIGH_SPEED` el sensor opera con un tiempo de muestreo de 20 ms, lo que puede resultar en errores de medición de hasta un 10 %, mientras que en modos más precisos, como `BETTER`, el tiempo de muestreo aumenta a 66 ms, reduciendo el error a cerca del 5 %. En esta aplicación, se utiliza un valor intermedio: un tiempo de muestreo de 33 ms, que se traduce en una precisión aproximada del 7 %, ofreciendo así un balance adecuado entre velocidad y exactitud [26]. El algoritmo de adquisición de datos empleado por los agentes utiliza la técnica de polling para obtener la información de los sensores a la máxima velocidad factible, lo que en ciertas condiciones puede superar el tiempo mínimo de muestreo recomendado, como ocurre

con el sensor VL53L0X, dando lugar a un caso de muestreo excesivo. , ocurre cuando un sensor es muestreado con una frecuencia mayor a la recomendada, lo que puede generar lecturas redundantes, incrementar el uso de recursos del sistema y del canal de comunicación y afectar la estabilidad de la adquisición de datos.

Esto resulta especialmente crítico en sistemas que implementan el bus I^2C(*Inter-Integrated Circuit*), donde el maestro debe esperar la confirmación (ACK) del esclavo tras cada petición antes de solicitar un nuevo dato. Un polling demasiado agresivo que omita esta espera puede provocar que el esclavo no procese a tiempo las transacciones, dando lugar a pérdida de datos o errores de sincronización [35].

Para evitar el muestreo excesivo del sensor de distancia VL53L0X, la biblioteca utilizada detiene la ejecución de todo el código hasta que el sensor responde con una medición nueva válida. Debido a que el algoritmo actual de adquisición de datos solicita una medición lo más rápido posible, el llamado a la función *get_distance()* no solo demora el tiempo necesario para leer la medición del sensor, sino que también incluye el tiempo que este requiere para adquirir y procesar un nuevo dato. Esto puede provocar tiempos de espera variables, que oscilan entre 23 y 33 ms.

Como solución, se propone modificar el algoritmo de adquisición de datos para evitar el bloqueo del sistema y optimizar la frecuencia de muestreo del sensor. Para ello, se implementa el uso de una variable denominada *flag_sensado*, que permite sincronizar el muestreo del sensor con el muestreo del controlador. De esta manera, no solo se asegura una frecuencia de muestreo adecuada para el controlador, sino que también se optimiza la respuesta del sistema al garantizar que cada medición sea utilizada en el cálculo de la actuación. Además, asegura que el tiempo de muestreo del algoritmo de control se mantenga en 40 ms, cumpliendo con el tiempo objetivo establecido para garantizar un desempeño óptimo del sistema. Esto contrasta con el algoritmo anterior, en el cual la adquisición de datos y la ejecución del control no estaban sincronizadas. Como resultado, la mitad de las mediciones obtenidas no se utilizaban en el cálculo de la actuación, lo que generaba una menor eficiencia en la respuesta del sistema.

A continuación, se presenta el enfoque de implementación en código utilizado:

```
def control_loop():
    if flag_sensado == 0: # Se requiere una nueva medicion
        medir_distancia() # Retorna la medicion en cm del sensor de distancia
        flag_sensado = 1 # Hay una medicion disponible que no se ha utilizado
        medir_velocidad() # Retorna la velocidad lineal en cm/s
        medir_orientacion() # Retorna el angulo del agente respecto de la linea central en radianes
    else:
        tiempo = t_actual - t_controlador
            if tiempo >= 0.04: # Tiempo de muestreo del control
                control_PID() # Calcula la actuacion en base a las mediciones de sensores
                flag_sensado = 0; # Se utilizo la medicion disponible
                t_controlador = t_actual
    actuacion_motores() # Aplica la actuacion calculada a los motores
```

Listing 3.1: Bucle de control en Python optimizado.

Este enfoque permite que tanto la adquisición de datos de los sensores como la ejecución del control PID funcionen de manera sincronizada, evitando bloqueos y asegurando la eficiencia del ciclo de control. Para evaluar el impacto de esta implementación, se realizó un nuevo

Tabla 3.2: Estadísticas de profiling relevantes en el ciclo de control

Función	N° llamados	Tiempo medio (s)	Tiempo acumulado (s)
`ciclo_de_control()`	197647	0.000515	101.8
`get_distance()`	2686	0.00308	8.273
`control_aux()`	2685	0	0.0008

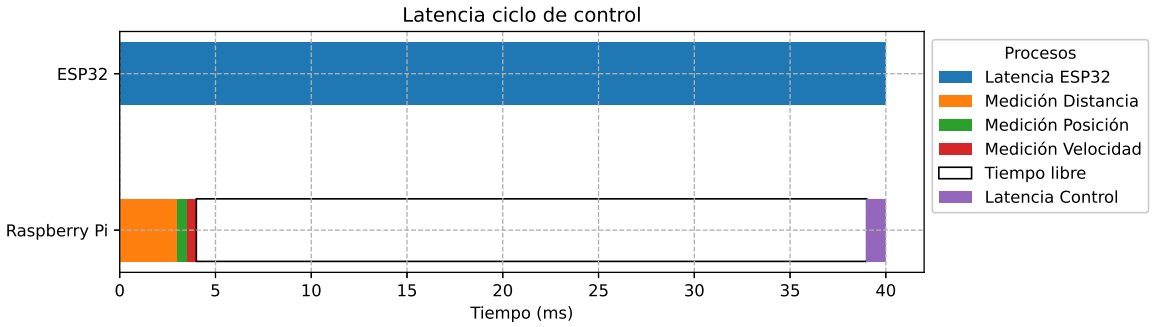


Figura 3.5: Tiempos de latencia ciclo de control optimizado

análisis de profiling con el fin de obtener las estadísticas de rendimiento mostradas en la Tabla 3.2.

De estos resultados se destaca que en cada ejecución del ciclo de control se realiza tanto la medición de distancia como la ejecución del control. Además, la función para obtener la medición de distancia ahora tiene un tiempo de respuesta de solo 3.08 ms, una disminución significativa en contraste con los tiempos obtenidos previamente. Esto evidencia que la implementación del algoritmo optimizó la adquisición de datos, permitiendo cumplir con el tiempo de muestreo objetivo de 40 ms y mejorando la eficiencia del sistema. La Figura 3.5 representa la latencia de la función del ciclo de control, donde se distingue un tiempo de holgura de aproximadamente 35 ms, el cual puede ser considerado para la implementación de algoritmos avanzados en futuros trabajos.

Es importante indicar que tanto el tiempo promedio como el número de llamados de la función *ciclo_de_control()* no deben considerarse para obtener conclusiones directas sobre el rendimiento del sistema. En la mayoría de estos llamados, no se cumple ninguna condición dentro del código, por lo que esta función solo actualiza el valor de la variable *tiempo*, sin afectar significativamente el procesamiento del control.

Los scripts optimizados y las modificaciones realizadas en el código fuente se encuentran disponibles en un repositorio público de GitHub, permitiendo su consulta y reutilización en futuros desarrollos de la plataforma.

4 | Experimentos y resultados

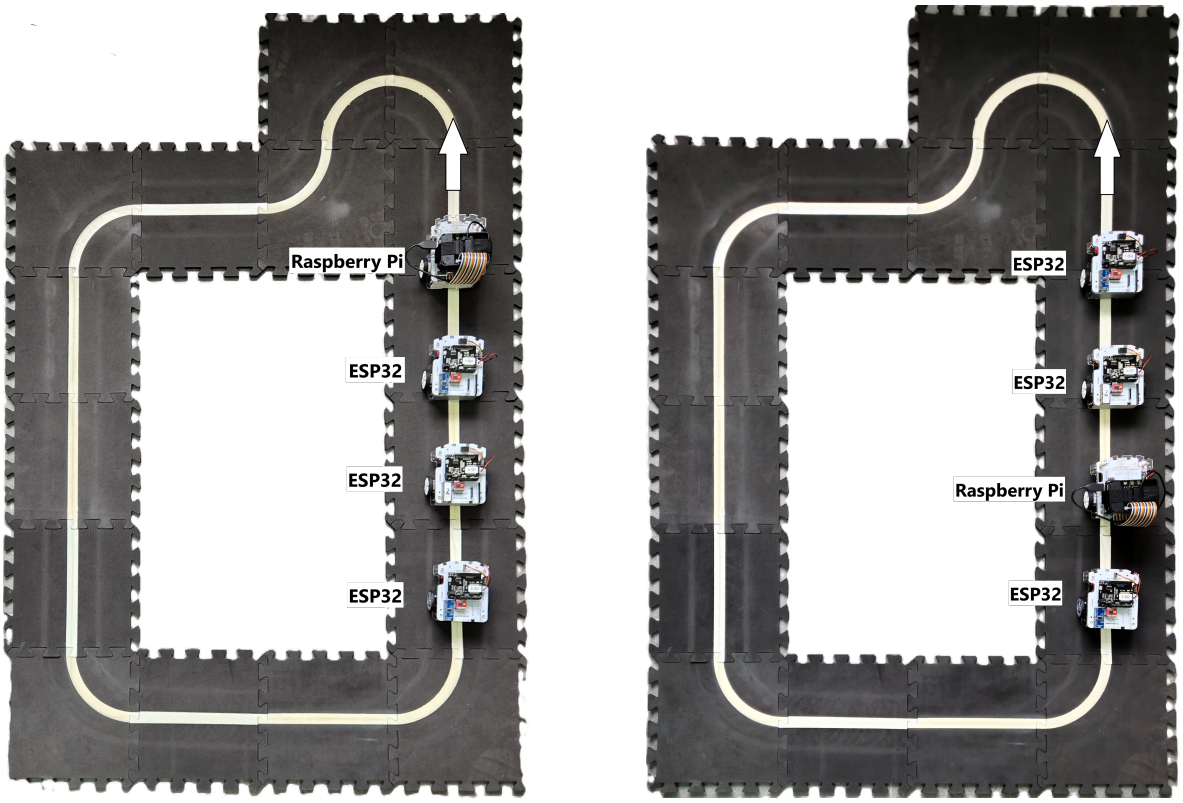
En este capítulo se describen las pruebas experimentales realizadas para evaluar y validar la integración de las mejoras implementadas previamente en el prototipo desarrollado, el cual, de manera preliminar, presenta un desempeño temporal similar al de los agentes basados en el microcontrolador ESP32.

Los experimentos realizados están enfocados en validar la operación de un pelotón al integrar este prototipo con Raspberry Pi, formando así un pelotón de agentes heterogéneo. Para ello, se utilizan tres agentes basados en ESP32 y un agente basado en Raspberry Pi, configurados para trabajar de manera coordinada. Para validar la integración de este último al pelotón, se consideraron dos versiones del experimento: en la primera, el agente con Raspberry Pi actúa como líder, según la configuración mostrada en la Figura 4.1a, con el objetivo de evaluar su capacidad para recibir instrucciones del monitor u operador a través de la interfaz gráfica de la plataforma, y transmitirlas al agente seguidor; en la segunda versión, el agente basado en Raspberry Pi ocupa el rol de segundo seguidor (tercer agente del pelotón), como se muestra en la Figura 4.1b, con el objetivo de validar su habilidad para recibir instrucciones de un agente seguidor que le precede y transmitir instrucciones a un agente sucesor.

Asimismo, se evaluó la repetibilidad del sistema en este contexto, garantizando que el comportamiento del pelotón permanezca consistente en múltiples ejecuciones bajo las mismas condiciones experimentales. Para validar la repetibilidad de los resultados obtenidos se aplicaron los mismos criterios de evaluación definidos en el Capítulo 2, es decir, evaluar de manera cuantitativa la consistencia de las distancias entre agentes y, de manera cualitativa, el seguimiento correcto del carril por parte de cada integrante del pelotón. Sin embargo, en este capítulo solo se reportan las métricas cuantitativas obtenidas a partir de las evaluaciones realizadas, la verificación cualitativa del desempeño del pelotón se puede revisar en el repositorio GitHub relacionado a este trabajo de memoria.

4.1. Metodología experimental

La metodología experimental sigue el mismo entorno de medición utilizado en experimentos previos, asegurando condiciones controladas para la evaluación del sistema. En esta ocasión, se emplea la pista mostrada en la Figura 4.2, la cual mantiene el mismo diseño que la utilizada en experimentos anteriores, pero con secciones rectas más largas. Esta modificación tiene como objetivo facilitar la identificación de dichas secciones en los datos adquiridos y proporcionar mayor tiempo a los algoritmos de control longitudinal para alcanzar la referencia de distancia establecida entre los agentes del pelotón. Los agentes tienen una separación inicial de 10 cm,



(a) Agente Raspberry Pi lider

(b) Agente Raspberry Pi tercer carro

Figura 4.1: Configuraciones de pelotón heterogéneo agentes ESP32 y Raspberry Pi

una distancia de referencia fija de 10 cm y una velocidad de referencia de 12 cm/s.

Durante el experimento, se registró la distancia medida por cada agente mientras completaba cuatro vueltas a la pista, y se realizaron tres iteraciones para evaluar la repetibilidad de los resultados. Estas cifras son menores en comparación con los experimentos previos realizados con un pelotón homogéneo de agentes basados en ESP32 debido a que la versión prototipo de Raspberry Pi utilizada tiene una menor autonomía, lo que limita la cantidad de iteraciones posibles. Además, la pista utilizada en este experimento es más larga, lo que incrementa el tiempo total de cada prueba y restringe el número de vueltas que pueden realizarse en una sesión experimental.

A los datos obtenidos se les aplica el mismo acondicionamiento señalado en los experimentos detallados en el Capítulo 2, obteniendo los gráficos de la Figura 4.3 para la primera serie de iteraciones con el agente Raspberry Pi como líder.

Las señales obtenidas presentan una forma similar a las registradas en experimentos previos y son consistentes con la pista utilizada. Visualmente, no se observan diferencias apreciables, y es posible distinguir con facilidad las vueltas realizadas por cada agente, esto sugiere que el comportamiento del sistema es consistente en múltiples ejecuciones, validando la estabilidad de la plataforma al integrar un agente basado en Raspberry Pi como líder en un pelotón heterogéneo.

Para cuantificar la variabilidad entre iteraciones y así evaluar la repetibilidad del sistema, se

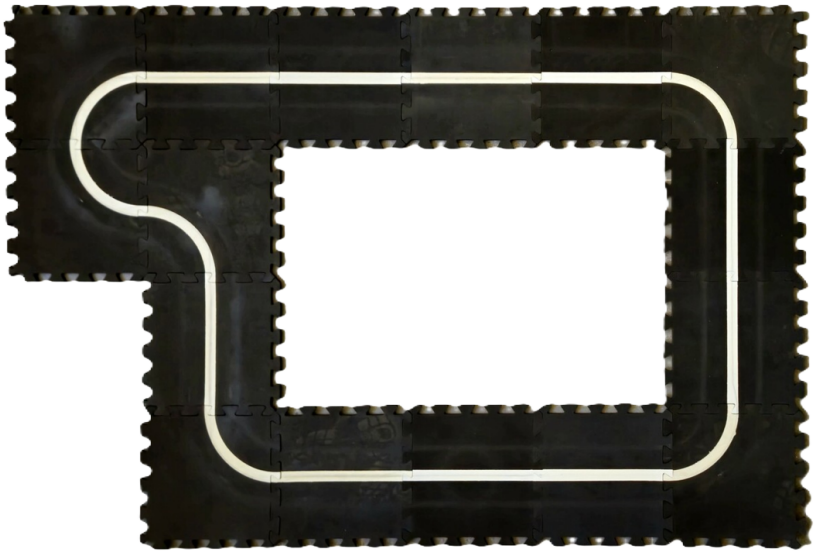


Figura 4.2: Pista utilizada en experimentos de repetibilidad con pelotón heterogéneo

Tabla 4.1: Resultados RMSE por iteración para cada carro

Carro	**Iteración 2**	**Iteración 3**
2do carro	3.0333 %	2.6399 %
3er carro	1.6283 %	1.6020 %
4to carro	2.5344 %	2.1234 %

utiliza el error RMSE normalizado según la definición detallada anteriormente en la Ecuación 2.2. Se toman los datos de la primera iteración como referencia y se obtienen los valores porcentuales mostrados en la Tabla 4.1.

En general, los valores obtenidos son superiores a los registrados en experimentos con un pelotón homogéneo de agentes ESP32, sin embargo, es importante señalar que estos valores son sensibles al acondicionamiento previo de las señales. No obstante, dado que la variabilidad se mantiene dentro de un margen aceptable, con desviaciones menores al 4 % respecto a la referencia establecida, se considera que el sistema sigue siendo repetible en este nuevo esquema de pelotón heterogéneo.

Por otra parte, al integrar el prototipo basado en Raspberry Pi como segundo seguidor en la plataforma según lo mostrado en la Figura 4.1b, se obtienen los resultados mostrados en los gráficos de la Figura 4.4.

Estas señales tienen forma similar a las mostradas en la Figura 4.3, lo que evidencia la consistencia de los datos en diferentes iteraciones del experimento. Se observa que el tercer carro, correspondiente al agente basado en Raspberry Pi, exhibe un desempeño más lento en el control longitudinal en comparación con los agentes basados en ESP32. Esto es previsible, dado que el modelo dinámico de esta nueva versión es distinto y requiere un ajuste más preciso de los

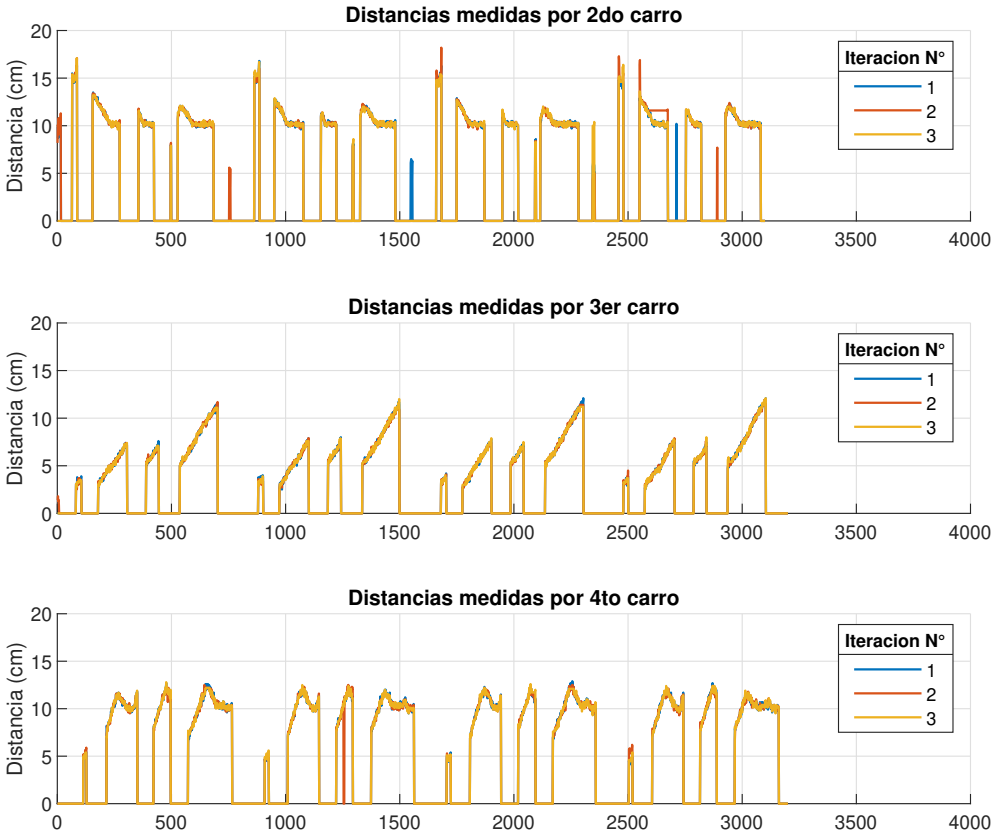


Figura 4.3: Resultados experimento repetibilidad con agente Raspberry Pi como líder

parámetros PID para mejorar su rendimiento. Sin embargo, estas diferencias no comprometen la estabilidad del pelotón, sino que únicamente afectan el tiempo de respuesta del agente. A partir de estos datos, se obtienen los valores de error RMSE normalizado presentados en la Tabla 4.2.

Los valores obtenidos respaldan la estabilidad general del sistema, evidenciando una baja variabilidad entre iteraciones. Aunque los errores registrados son mayores en comparación con un pelotón homogéneo de ESP32, la desviación observada se mantiene dentro de un rango aceptable, (menores al 5 %) lo que sugiere que el sistema sigue siendo repetible en un entorno heterogéneo.

Las diferencias en los valores de RMSE no afectan significativamente el desempeño global del pelotón, lo que confirma la viabilidad de integrar el agente basado en Raspberry Pi sin comprometer la estabilidad de la plataforma.

A partir de los resultados obtenidos y las mejoras implementadas, se confirma que la nueva versión de los agentes de la plataforma RUPU basados en Raspberry Pi se integra de manera efectiva, permitiendo la formación de un pelotón heterogéneo. Los resultados muestran que esta versión mantiene una operación estable dentro del pelotón y genera datos consistentes, validando así la efectividad de las mejoras aplicadas.

En general, los resultados ratifican la estabilidad del sistema y evidencian que la incorporación

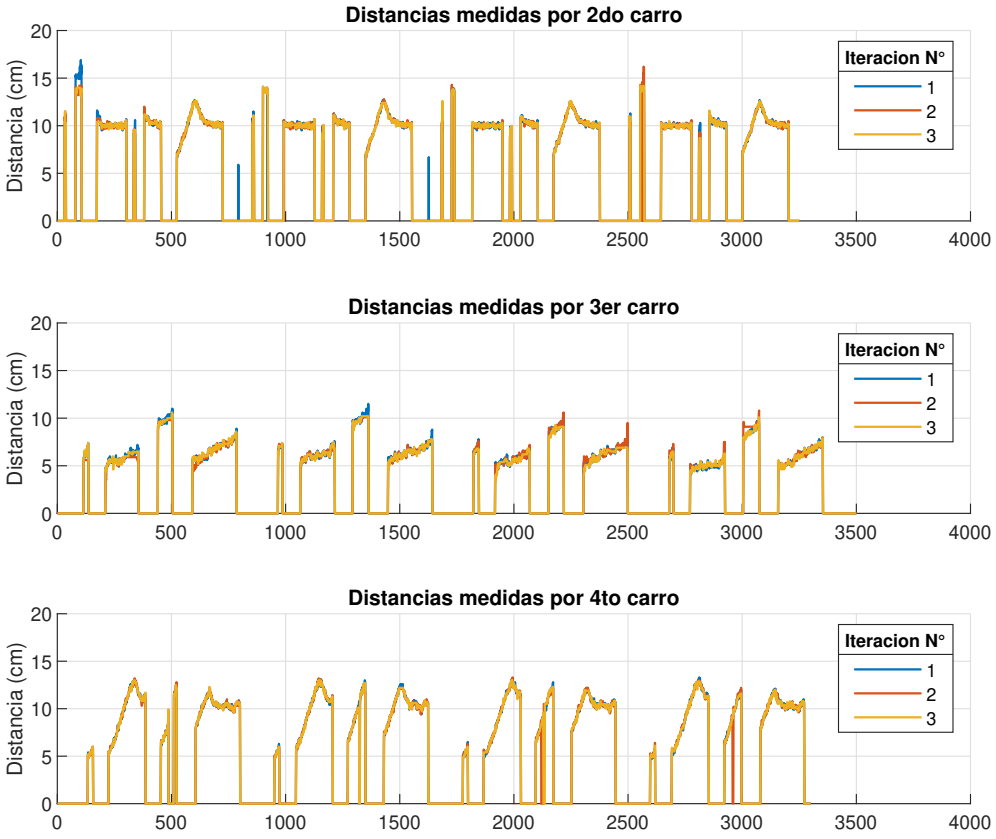


Figura 4.4: Resultados experimento repetibilidad con agente Raspberry como 3er carro en pelotón mixto

de agentes heterogéneos en la plataforma RUPU no compromete su desempeño. Estos hallazgos abren la posibilidad de futuras optimizaciones en la sintonización de controladores y en el procesamiento de datos, con el fin de incrementar la precisión y eficiencia del pelotón heterogéneo.

Los scripts utilizados para la ejecución y análisis de los experimentos están disponibles en el repositorio de GitHub, asegurando la transparencia y futura replicabilidad de los resultados.

Tabla 4.2: Resultados RMSE por iteración para cada carro

Carro	**Iteración 2**	**Iteración 3**
2do carro	2.6154 %	2.7194 %
3er carro	4.5688 %	3.2983 %
4to carro	2.1596 %	1.7828 %

5 | Conclusiones y trabajo futuro

Este capítulo presenta las conclusiones más relevantes derivadas de los ensayos cuantitativos expuestos en el capítulo 4; sin embargo, la incorporación exitosa de la nueva versión de los agentes en un pelotón heterogéneo no se basa únicamente en esos resultados numéricos, sino también en las pruebas cualitativas individuales que hicieron posible validar e implementar cada mejora. La sección de conclusiones resume los hallazgos más sobresalientes y evalúa su repercusión en la repetibilidad, usabilidad e interoperabilidad del sistema, mientras que la sección de trabajo futuro propone las líneas de investigación y refinamientos necesarios para reforzar la estabilidad, la eficiencia energética y la escalabilidad de la plataforma.

5.1. Conclusiones

Este trabajo ha abordado la optimización y validación de la plataforma experimental RUPU, enfocándose en la integración de agentes basados en Raspberry Pi para ampliar sus capacidades y mejorar su usabilidad y la repetibilidad de los experimentos. Se realizaron diversas mejoras en la infraestructura del sistema, incluyendo la optimización del protocolo de comunicación, la reestructuración del código, el rediseño de componentes y la evaluación experimental del desempeño de los agentes en formaciones de pelotón.

Uno de los aspectos más relevantes evaluados en este estudio fue la repetibilidad de los experimentos, siguiendo los criterios establecidos por la *Association for Computing Machinery* (ACM). Se realizaron múltiples iteraciones del mismo experimento con los agentes basados en ESP32 bajo condiciones controladas, observando que las mediciones obtenidas mantuvieron una correlación significativa entre repeticiones. La baja variabilidad en los resultados obtenidos indica que la plataforma puede generar datos consistentes, validando su uso como herramienta experimental confiable para el análisis de algoritmos de control distribuido.

Se implementaron mejoras importantes en diversos aspectos del sistema. Se utilizaron herramientas estadísticas para optimizar los algoritmos de adquisición de datos de los sensores, mejorando la precisión y estabilidad de las mediciones, lo que permitió reducir los tiempos de latencia del ciclo de control. Asimismo, se identificaron sensores inadecuados para ciertas tareas específicas y se realizó un estudio para su posible reemplazo con alternativas más eficientes. Paralelamente, se llevó a cabo un rediseño estructural que incluyó modificaciones en la PCB, la fuente de poder y la disposición de los componentes, optimizando la integración de hardware y facilitando el mantenimiento del sistema. En este rediseño se tomó en cuenta la necesidad de desarrollar una nueva versión que permita implementar mejoras futuras sin requerir modificaciones significativas, asegurando la escalabilidad y flexibilidad de la plataforma.

Finalmente, se evaluó la interoperabilidad entre agentes basados en ESP32 y Raspberry Pi dentro de un mismo pelotón. Los resultados experimentales mostraron que la integración es factible, permitiendo la formación de pelotones heterogéneos con una sincronización adecuada. Sin embargo, se identificaron diferencias en el tiempo de respuesta y desempeño entre ambas versiones, lo que sugiere la necesidad de ajustes adicionales en los parámetros de control.

A partir del trabajo realizado se pueden rescatar importantes consideraciones para el diseño de una plataforma experimental de control distribuido. En el ámbito de hardware es fundamental seleccionar componentes que satisfagan los requisitos de discretización estrictos del control; además de su frecuencia de muestreo nominal, conviene evaluar los sobrecargos introducidos por el protocolo de comunicación y el procesamiento de señales. Por ello, es recomendable optar por dispositivos de fabricantes reconocidos o someterlos a pruebas exhaustivas que confirmen que sus parámetros reales coinciden con los teóricos especificados. También es necesario diseñar una PCB que ofrezca la flexibilidad propia de un dispositivo orientado a la investigación, integrando todos los componentes y permitiendo añadir nuevos elementos para mejoras continuas sin rediseños estructurales.

En el ámbito de software, resulta esencial evaluar el impacto de las bibliotecas y frameworks en el tratamiento de datos, seleccionar el método de interacción con los dispositivos (polling o interrupciones) según los requisitos de tiempo de respuesta, considerar la implementación de protocolos de comunicación compatibles con lo definido para el resto de componentes asociados a la plataforma y asegurar la correcta configuración de los componentes externos para garantizar su funcionamiento óptimo en cuanto a tiempo de respuesta, precisión de los datos entregados y eficiencia energética. Estas consideraciones están orientadas al desarrollo de un dispositivo funcional capaz de entregar resultados repetibles.

Todo el código fuente desarrollado en este trabajo, así como la documentación asociada, ha sido publicado en un repositorio de GitHub, con el objetivo de facilitar su uso en futuras investigaciones y contribuir a la mejora continua de la plataforma RUPU.

5.2. Trabajo Futuro

A pesar de los avances logrados, aún existen áreas de mejora, especialmente en la optimización del control tanto longitudinal como lateral y en la eficiencia energética de los agentes basados en Raspberry Pi. Refinar estos aspectos permitirá mejorar la estabilidad y autonomía de la plataforma, fortaleciendo su desempeño en escenarios experimentales más exigentes.

El desarrollo de la plataforma RUPU continúa en evolución, con un gran potencial para consolidarse como una herramienta de referencia en la investigación de sistemas de control distribuido. A partir de los hallazgos de este estudio, se proponen las siguientes líneas de trabajo futuro:

- **Explorar protocolos de comunicación**: Aunque la implementación de UDP permitió la interoperabilidad entre agentes heterogéneos, se recomienda investigar alternativas más robustas que ofrezcan un mayor nivel de calidad de servicio. Opciones como UDP con control de errores, UTP o protocolos con menor latencia podrían mejorar la estabilidad de la comunicación en escenarios con múltiples agentes y facilitar el análisis de datos posterior.

- **Automatización de los experimentos**: El acondicionamiento de las señales aplicado altera las mediciones obtenidas por lo que los resultados finales, aunque son suficientes para obtener conclusiones generales, no son completamente representativos del desempeño del sistema. Esto ocurre debido a un desfase temporal introducido por la intervención manual del operador y la pérdida de paquetes en la comunicación. Para mitigar este problema, se recomienda automatizar la ejecución de los experimentos, minimizando la intervención humana. Además, sería beneficioso incorporar una variable temporal que permita sincronizar los datos entre experimentos, asegurando métricas más precisas y representativas del desempeño de la plataforma.
- **Estudio energético**: Hasta ahora, la evaluación del sistema se ha enfocado en pruebas cualitativas y en tiempos de latencia de los algoritmos y respuesta del control. Sin embargo, dado que se trabaja con robots autónomos, se sugiere incluir métricas de consumo energético para evaluar la eficiencia de futuras mejoras. También se recomienda estudiar distintos esquemas de control y su impacto en el consumo de energía con el fin de optimizar la autonomía de los agentes.
- **Integración de visión por computador**: Uno de los objetivos de la versión basada en Raspberry Pi es la implementación de algoritmos de visión por computador para el seguimiento de carril. Se recomienda profundizar en esta línea de investigación para desarrollar sistemas de navegación más avanzados que no solo mejoren el control lateral, sino que también contribuyan al control longitudinal y al reconocimiento de curvas. Esto ampliaría significativamente las aplicaciones de la plataforma y permitiría una mayor autonomía en la toma de decisiones de los agentes.
- **GUI integrada en agentes**: Actualmente, la interfaz de usuario de la plataforma está basada en Python, lo que requiere un dispositivo externo con un intérprete compatible y las bibliotecas necesarias para su correcto funcionamiento. Se recomienda evaluar la integración de la interfaz en un servidor alojado en los agentes basados en Raspberry Pi, permitiendo así su acceso a través de una interfaz web. De este modo, cualquier dispositivo con un navegador compatible podrá ejecutar las mismas funciones sin depender de un sistema operativo específico ni de bibliotecas adicionales.
- **Exploración de diferentes estrategias de control**: Uno de los resultados de este trabajo fue tener un tiempo de holgura en la latencia del ciclo de control superior a 30 ms (Revisar Figura 3.5), lo cual permite investigar la implementación de estrategias de control avanzadas que exijan mayor capacidad de cómputo (como algoritmos de estimación y técnicas de optimización de parámetros), ajustar el tiempo de muestreo para acelerar el ciclo de control o para tener un tiempo de holgura mayor y evaluar de manera sistemática el impacto de estas modificaciones en la estabilidad y el rendimiento global de los agentes.

6 | Anexos

En esta sección se detalla el desarrollo de las mejoras propuestas en 2 que no fueron relevantes para la ejecución de los experimentos de repetibilidad de la nueva versión de los agentes basados en Raspberry Pi. Estas mejoras están enfocadas en aumentar la usabilidad de la plataforma

6.1. Rediseño de la estructura de los agentes

Tras la optimización de los algoritmos de adquisición de datos y el desarrollo de un prototipo funcionalmente similar a los agentes basados en ESP32, se dio inicio a la implementación de mejoras estructurales. Estas mejoras se enfocan en integrar la Raspberry Pi de manera eficiente junto con una fuente de alimentación adecuada, asegurando compatibilidad con las optimizaciones previas.

En trabajos previos, se identificaron problemas en la integración de una cámara en los agentes basados en Raspberry Pi, especialmente en la detección del carril en curvas. La principal hipótesis es que el modelo cinemático actual no es compatible con la integración de una cámara frontal en una plataforma DDMR (Differential Drive Mobile Robot). Ante esta problemática, se consideró necesario reevaluar el modelo cinemático de los agentes, analizando también su impacto en las mediciones de los arreglos de sensores QTR inferiores. Para ello, se estudiaron plataformas similares de tipo DDMR que integran una cámara. Entre ellas, se identificó la plataforma JetBot de NVIDIA, un robot basado en Jetson Nano diseñado para el desarrollo de aplicaciones de visión artificial y aprendizaje profundo. JetBot, una plataforma compacta y de código abierto, permite la integración de modelos de IA para navegación autónoma. Su diseño ha servido como referencia para evaluar modificaciones en el modelo cinemático de los agentes RUPU, facilitando la optimización de sistemas de visión en robots móviles. En contraste con los agentes de la plataforma RUPU, este robot también con tracción diferencial tiene el centro de giro adelantado junto con una cámara frontal para el seguimiento de carril, como se muestra en la Figura 6.1. Esta plataforma, junto con la documentación asociada, permiten definir criterios preliminares para evaluar la modificación del modelo cinemático de los agentes RUPU.

Se obtuvieron las siguientes conclusiones relativas a la ubicación del centro de giro de los agentes y por lo tanto de su modelo cinemático considerando el uso de cámara y arreglo de sensores QTR:

- Si el centro de giro se encuentra hacia la parte trasera, los sensores QTR serán más sensibles a cambios en el giro del robot, lo que permitirá un control más preciso y estable

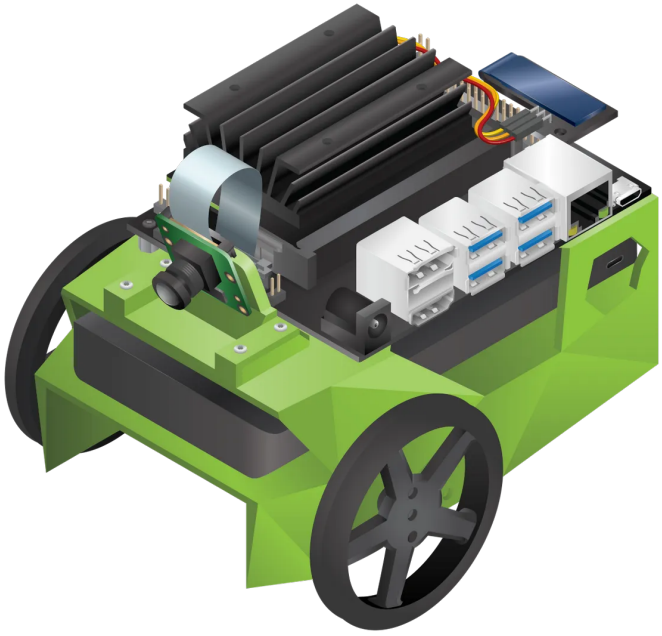


Figura 6.1: Robot JetBot

en el seguimiento de líneas. No obstante, en esta configuración, la cámara frontal será más sensible a pequeños cambios en la orientación, ya que el arco de la curvatura de su movimiento será mayor, lo que podría hacer que pierda con más facilidad el horizonte cercano.

- Si el centro de giro está más adelante, el giro de la cámara será proporcional al giro del robot, permitiendo mantener mejor el horizonte cercano. Sin embargo, los sensores QTR situados en la parte delantera requerirán cambios más notorios en la trayectoria para detectar desviaciones en la línea, lo que podría afectar la estabilidad del control.

Ya que actualmente los agentes utilizan sólo el arreglo de sensores QTR para el seguimiento de carril, se considera más adecuado mantener el modelo cinemático, es decir, un enfoque con el centro de giro posterior considerando un balance entre estabilidad, maniobrabilidad y capacidad de respuesta ante variaciones en el entorno. No obstante, se tendrá en consideración desarrollar la estructura lo suficientemente flexible para poder integrar visión por computador mediante cámara sin mayores problemas.

6.1.1. Fuente de poder para la nueva versión de los agentes

La implementación de una nueva fuente de poder es fundamental para mejorar la autonomía y confiabilidad de los robots. En caso de utilizar múltiples celdas, se considerará la integración de un sistema de administración de energía. Además, se tendrá en cuenta la disponibilidad en el mercado de la opción escogida para facilitar su reparación o reemplazo en el futuro.

Es fundamental integrar una fuente de poder que cumpla con los requerimientos de potencia de la placa utilizada. Actualmente, se emplea una Raspberry Pi 4, la cual requiere una fuente



Figura 6.2: Fuente de Poder UPS 3S

capaz de suministrar 5V y 3A mediante un conector USB-C. Aunque esta versión sigue vigente, la más reciente, Raspberry Pi 5, demanda una fuente de 5V y 5A con el mismo tipo de conector.

El diseño de la nueva versión de los agentes debe ser flexible, permitiendo la integración de futuras mejoras sin comprometer su funcionalidad actual. Por ello, la selección de la fuente de poder considerará los requerimientos de la Raspberry Pi 5, garantizando compatibilidad con versiones más avanzadas y proporcionando una base estable para futuras actualizaciones del sistema.

Para satisfacer estos requerimientos, se ha seleccionado el módulo de alimentación **UPS Module 3S** de *Waveshare* [36] mostrado en la Figura 6.2. Este módulo ofrece las siguientes características clave:

- Soporta baterías de litio 18650 en configuración 3S, un formato estándar disponible en múltiples distribuidores, lo que garantiza una alimentación estable y una mayor capacidad energética.
- Proporciona una salida de **5V** con una corriente de hasta **5A** a través de un conector **USB-C**, adecuada para alimentar la Raspberry Pi 4 y 5.
- Incorpora un sistema de administración de energía con protección contra sobrecarga y sobredescarga, garantizando una operación segura y eficiente. A diferencia de las baterías utilizadas en los agentes basados en ESP32, este módulo posee un cargador incorporado, lo que facilita su uso y mantenimiento. Además, optimiza el rendimiento de las celdas, prolongando su vida útil.
- Dispone de monitoreo en tiempo real del voltaje, la corriente y el estado de carga a través del sensor INA219, lo que facilita la gestión de energía del sistema.

Gracias a estas características, el UPS Module 3S se adapta de manera óptima a la nueva versión de los agentes, garantizando un suministro de energía estable y escalable para futuras mejoras sin comprometer la confiabilidad del sistema. Además, el monitoreo en tiempo real de las variables de potencia puede utilizarse como una métrica relevante para evaluar el desempeño de diferentes algoritmos o la integración de nuevos componentes, permitiendo un análisis más

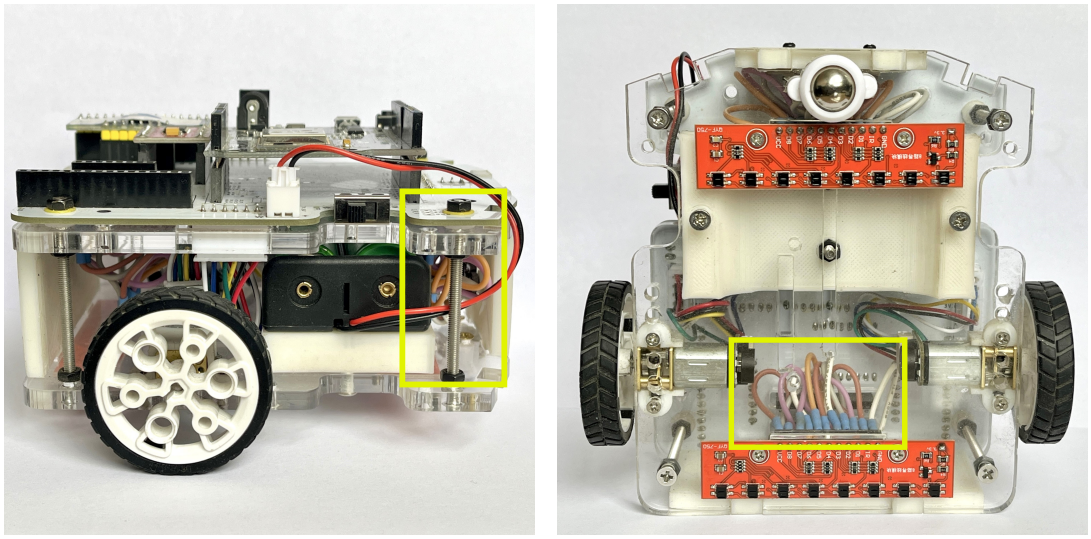


Figura 6.3: Agentes RUPU destacando la conexión por cable de sensores

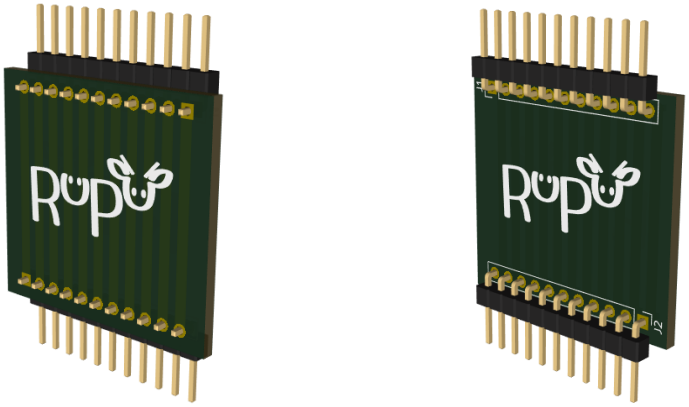


Figura 6.4: Propuesta de PCB reemplazo conector

detallado de la eficiencia energética de estos en el sistema.

6.1.2. Diseño de PCB

En el diseño actual de los agentes, la conexión entre los sensores QTR inferiores con la PCB principal es a través de cables cuyos conectores deben ser armados de forma manual durante el proceso de ensamblado de la plataforma; este enfoque entrega un resultado desordenado y propenso a errores de armado, como se muestra marcado en amarillo en la Figura 6.3, además, requieren de un tiempo considerable para su ensamblaje.

Se propone reemplazar esta conexión diseñando una PCB apropiada que sirva como un conector rígido, que pueda incluir los logotipos diseñados en [2] y se pueda integrar tanto en la versión de los agentes basada en ESP32 como en Raspberry Pi. Este diseño se muestra en la Figura 6.4.

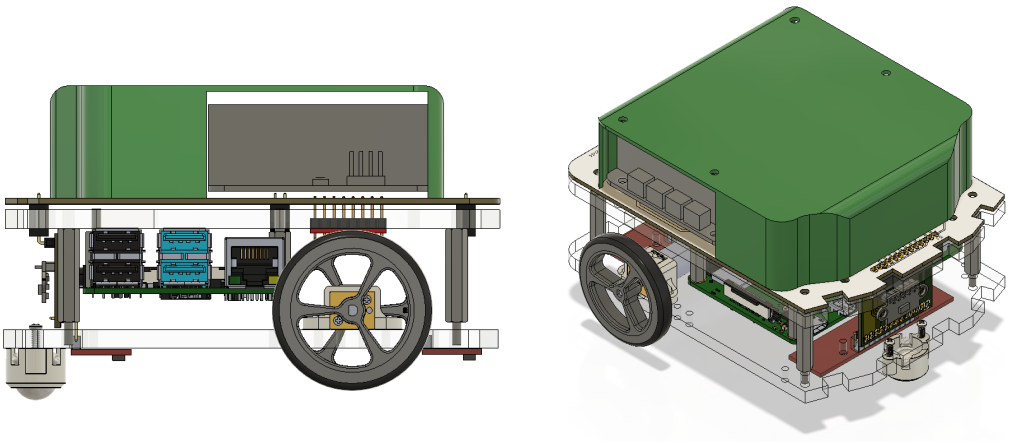


Figura 6.5: Versión final nueva versión de agente RUPU Raspberry Pi

Por otra parte, se considera crítica la correcta integración de la Raspberry Pi en los agentes, por lo que se plantea el diseño de una PCB basada en la especificación de un HAT (Hardware Attached on Top). Este diseño facilitará una integración modular y eficiente de sensores y actuadores con la Raspberry Pi, eliminando la necesidad de cableado externo. Para este diseño, se tomarán como referencia las especificaciones oficiales de HAT de Raspberry Pi [37,38].

Se contempla mantener en lo posible la estética del prototipo desarrollado, ya que integra los mismos actuadores y sensores en su estructura actual. Esto garantizará una transición sin afectar la compatibilidad de hardware y facilitará la adopción de la nueva PCB sin necesidad de rediseñar completamente la carcasa o la disposición interna de los componentes, además de considerar la integración óptima de las mejoras de hardware ya implementadas y probadas. Otro aspecto clave en el diseño de la PCB es que pueda integrar de manera óptima la fuente de poder seleccionada y la alimentación a los diferentes componentes.

Las mejoras implementadas en los agentes optimizan su estructura, gestión de energía y capacidad de procesamiento. La integración de la Raspberry Pi y el diseño de PCB basado en un HAT facilitan la conexión de sensores y actuadores, reduciendo el cableado y mejorando la modularidad. La fuente de alimentación UPS Module 3S proporciona estabilidad energética y monitoreo en tiempo real, mientras que el ADC MCP3208 mejora la adquisición de datos analógicos. La Figura 6.5 muestra un plano lateral e isométrico de la nueva versión de los agentes, resaltando los cambios implementados. Además, se incorpora una carcasa que protege y oculta las conexiones, proporcionando un acabado más prolijo. Esta carcasa también sirve como soporte para la cámara y permite su rediseño para optimizar la posición de esta en el seguimiento de carril. Estas mejoras consolidan una plataforma más eficiente y adaptable a futuras optimizaciones.

6.2. Pruebas de desempeño Python y C

Como se señaló y detalló en el capítulo 3, la función principal que se ejecuta durante la operación de los agentes es *ciclo_de_control()*, cuyas tareas fundamentales son la adquisición de datos y la aplicación del algoritmo de control para calcular la actuación de los motores. La latencia de las funciones relacionadas con la adquisición de datos de los sensores depende

Tabla 6.1: Comparativa desempeño algoritmo de control en Python y C

Lenguaje de programación	Utilización CPU (%)	Memoria Usada (MB)	Tiempo medio de cómputo (ms)	Tiempo total de cómputo (ms)
Python	105.63	22.05	0.017	8.274
C	54.02	1.38	0.008	3.950

principalmente de la frecuencia de muestreo de estos, mientras que la latencia del cálculo del control PID está condicionada por el tiempo de cómputo en la Raspberry Pi, considerando el overhead intrínseco del lenguaje de programación utilizado. En esta versión de los agentes, los algoritmos están implementados completamente en Python, un lenguaje interpretado que impone requerimientos de tiempo de procesamiento superiores a los de los lenguajes compilados.

Con el objetivo de analizar una posible mejora que amplíe el tiempo de holgura y contribuya a la investigación de otras estrategias de control con mayor carga computacional, se aisló el algoritmo de control PID en cascada y se diseñaron pruebas comparativas entre implementaciones en Python y en C. En cada ensayo se registraron los tiempos de cómputo por ciclo de control y el uso de recursos de la Raspberry Pi para cuantificar las diferencias en latencia.

Para la implementación de estos algoritmos se consideraron los mismos parámetros y valores de referencia, entradas con forma exponencial que convergen a los valores de referencia para simular el desempeño dinámico de una planta estable, una frecuencia de muestreo de 100 Hz y 500 iteraciones, es decir, una simulación de 5 segundos, obteniendo las métricas mostradas en la Tabla 6.1

Las métricas de tiempo de cómputo, uso de CPU y consumo de memoria se obtuvieron de manera análoga en ambas implementaciones. El tiempo medio y total de cómputo se midió registrando el instante antes y después de cada iteración del bucle de simulación: en C con `clock_gettime(CLOCK_MONOTONIC,...)` y en Python con `time.perf_counter()`, calculando la diferencia y convirtiéndola a milisegundos. El uso de CPU por núcleo se determinó usando `getrusage(RUSAGE_SELF)` para extraer el tiempo de usuario y sistema (user+sys) antes y después de cada iteración, y dividiendo ese delta de CPU-time entre el tiempo transcurrido. El pico de memoria se obtuvo leyendo la línea `VmRSS` de `/proc/self/status` (fichero virtual del sistema de archivos `procfs` que el kernel expone en memoria para describir el estado del proceso actual).

Cabe destacar que en Python el uso de CPU puede exceder el 100 % debido a la sobrecarga del intérprete y de bibliotecas nativas (por ejemplo, NumPy y las llamadas a `getrusage`), que a menudo distribuyen cómputo en múltiples hilos y se contabilizan como carga sobre varios núcleos. Esto confirma que, aunque Python facilita un desarrollo más ágil, la versión en C aporta una latencia inferior, un menor uso de CPU y memoria, aspectos críticos para aplicaciones de control en tiempo real en sistemas embebidos como la Raspberry Pi, y representa un área de mejora que podría permitir la implementación de algoritmos de control más complejos y sensores avanzados sin comprometer los recursos disponibles.

6.3. Consideraciones sobre el uso de la nueva versión

La implementación de la nueva versión de los agentes en la plataforma requiere ciertas consideraciones para garantizar una transición eficiente y mejorar la usabilidad del sistema. En esta sección, se detallan los aspectos clave a tener en cuenta, incluyendo la migración de código y bibliotecas debido a la actualización del sistema operativo, así como la automatización de la ejecución de los scripts para facilitar su operación.

6.3.1. Actualización de sistema operativo y migración de código para bibliotecas compatibles

Una de las mejoras propuestas en el Capítulo 2 está relacionada con cambiar la topología de la red para evitar el uso de un componente externo que administre y configure esta. Para ello, se propone configurar la Raspberry Pi como HotSpot, reemplazando así el componente externo (Router). Para implementar esta mejora, es conveniente actualizar el Sistema Operativo a la última versión disponible, que corresponde a Debian 12, una distribución de Linux para Raspberry Pi que incorpora un administrador de redes que facilita el proceso de configuración para punto de acceso.

Al actualizar el sistema operativo, también es necesario revisar aquellas bibliotecas que no son compatibles y buscar reemplazos equivalentes. En particular, la biblioteca que permite el manejo de los pines GPIO de la Raspberry Pi debió ser actualizada por una versión de bajo nivel, lo que requirió modificar el código para migrar al uso de esta nueva biblioteca. Cabe señalar que esta versión del sistema operativo trabaja con entornos virtuales para los scripts de Python, lo que ofrece varias ventajas, tales como:

- **Aislamiento de dependencias**: Permite que cada proyecto utilice versiones específicas de bibliotecas sin interferencias con otros proyectos.
- **Reproducibilidad**: Garantiza que los scripts se ejecuten en un entorno controlado, evitando problemas con actualizaciones de bibliotecas del sistema.
- **Facilidad de despliegue**: Se pueden replicar entornos de manera sencilla en diferentes dispositivos sin necesidad de instalar globalmente cada dependencia.

6.3.2. Ejecución automatizada y parametrizada del script de Python

Para realizar la configuración inicial, donde se especifican los puertos e IPs relativos a la comunicación y la etiqueta de cada robot a los agentes basados en ESP32, se deben conectar por USB y modificar el código utilizando el IDE de Arduino. En contraste, dado que la Raspberry Pi es en principio un computador, este procedimiento es más complejo. Para realizar esta configuración inicial y modificar las variables respectivas en el script, se requiere contar con los periféricos necesarios, como monitor, mouse y teclado, o modificar el script de manera remota a través de una sesión SSH utilizando una terminal apropiada.

Posteriormente, es necesario activar el entorno virtual y habilitar los servicios requeridos para la correcta operación del código, para finalmente ejecutarlo mediante comandos. Con el objetivo de facilitar este procedimiento y mejorar la usabilidad de esta versión de los agentes,

Tabla 6.2: Comandos de configuración inicial

Comando	**Descripción**
C/label/valor	Define la etiqueta del robot; “L” para líder
C/ip_mon/valor	Define la dirección IP de monitor
C/ip_seg/valor	Define la dirección IP de seguidor

se preparó un script bash que automatiza este proceso y ejecuta el código principal cada vez que se encienda la placa.

Además, se modificó el script para recibir los parámetros de configuración inicial de manera remota, utilizando la misma estructura de mensajes mostrada en 2.1, implementada en los comandos mostrados en la Tabla 6.2 . Esto permite mayor flexibilidad en la configuración y reduce la necesidad de intervención manual en cada dispositivo.

Bibliografía

[1] C. Escobar, "Diseño, implementación y evaluación de una plataforma experimental para el estudio del control de una flota de robots móviles," 2021.

[2] C. Chaufleur, "Diseño e implementación de interfaz de usuario y casos de uso para plataforma experimental de sistema multi-agentes," 2023.

[3] P. Castillo, "Integración de sistema de visión por computador a plataforma robótica móvil con tracción diferencial," 2022.

[4] J. Guerrero-Ibáñez, S. Zeadally, and J. Contreras-Castillo, "Sensor technologies for intelligent transportation systems," *Sensors*, vol. 18, no. 4, p. 1212, 2018.

[5] K. C. Dey, L. Yan, X. Wang, Y. Wang, H. Shen, M. Chowdhury, L. Yu, C. Qiu, and V. Soundararaj, "A review of communication, driver characteristics, and controls aspects of cooperative adaptive cruise control (cacc)," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 2, pp. 491–509, 2015.

[6] K. Bengler, K. Dietmayer, B. Farber, M. Maurer, C. Stiller, and H. Winner, "Three decades of driver assistance systems: Review and future perspectives," *IEEE Intelligent transportation systems magazine*, vol. 6, no. 4, pp. 6–22, 2014.

[7] V. Turri, B. Besselink, and K. H. Johansson, "Cooperative look-ahead control for fuel-efficient and safe heavy-duty vehicle platooning," *IEEE Trans. on Control Systems Technology*, vol. 25, no. 1, pp. 12–28, 2016.

[8] L. Xu, L. Y. Wang, G. Yin, and H. Zhang, "Communication information structures and contents for enhanced safety of highway vehicle platoons," *IEEE Trans. on Vehicular Technology*, vol. 63, no. 9, pp. 4206–4220, 2014.

[9] S. Thormann, A. Schirrer, and S. Jakubek, "Safe and efficient cooperative platooning," *IEEE Trans. on Intelligent Transportation Systems*, 2020.

[10] A. I. Maass, F. J. Vargas, and A. A. Peters, "String stability for predecessor-following platoons over channels with heterogeneous reception probabilities*," in *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pp. 927–933, 2024.

[11] C. Bergenhem, S. Shladover, E. Coelingh, C. Englund, and S. Tsugawa, "Overview of platooning systems," in *Proceedings of the 19th ITS World Congress, Oct 22-26, Vienna, Austria (2012)*, 2012.

[12] C. Latrech, A. Chaibet, M. Boukhnifer, and S. Glaser, "Integrated longitudinal and lateral networked control system design for vehicle platooning," *Sensors*, vol. 18, no. 9, 2018.

[13] S. Wei, Y. Zou, X. Zhang, T. Zhang, and X. Li, "An integrated longitudinal and lateral vehicle following control system with radar and vehicle-to-vehicle communication," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1116–1127, 2019.

[14] A. Bayuwindra, J. Ploeg, E. Lefeber, and H. Nijmeijer, "Combined longitudinal and lateral control of car-like vehicle platooning with extended look-ahead," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 3, pp. 790–803, 2020.

[15] L. Yu, Y. Bai, Z. Kuang, C. Liu, and H. Jiao, "Intelligent bus platoon lateral and longitudinal control method based on finite-time sliding mode," *Sensors*, vol. 22, no. 9, p. 3139, 2022.

[16] F. A. Papoulias, "Guidance and control laws for vehicle pathkeeping along curved trajectories," *Applied ocean research*, vol. 14, no. 5, pp. 291–302, 1992.

[17] T. I. Fossen, K. Y. Pettersen, and R. Galeazzi, "Line-of-sight path following for dubins paths with adaptive sideslip compensation of drift forces," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 2, pp. 820–827, 2014.

[18] A. Roy and M. M. Noel, "Design of a high-speed line following robot that smoothly follows tight curves," *Computers & Electrical Engineering*, vol. 56, pp. 732–747, 2016.

[19] M. Velasco-Villa, R. D. Cruz-Morales, A. Rodriguez-Angeles, and C. A. Domínguez-Ortega, "Observer-based time-variant spacing policy for a platoon of non-holonomic mobile robots," *Sensors*, vol. 21, no. 11, p. 3824, 2021.

[20] "Ensemble." `https://platooningensemble.eu`.

[21] "Assembly - duckiebot." `https://docs.duckietown.com/daffy/opmanual-duckiebot/assembly/db21j/index.html`.

[22] C. Escobar, F. J. Vargas, A. A. Peters, and G. Carvajal, "A cooperative control algorithm for line and predecessor following platoons subject to unreliable distance measurements," *Mathematics*, vol. 11, no. 4, 2023.

[23] Association for Computing Machinery, "Artifact review and badging." `https://www.acm.org/publications/policies/artifact-review-badging`, Aug. 2020.

[24] "Git elo308_rupu." `https://github.com/FRifoM99/ELO308_RUPU`.

[25] F. Rifo, N. Salvador, F. Donoso, A. Peters, G. Carvajal, and F. Vargas, "Experimental sensitivity analysis of a control algorithm for line following platooning," in *2024 13th International Workshop on Robot Motion and Control (RoMoCo)*, pp. 43–48, 2024.

[26] "Vl53l0x time-of-flight (tof) ranging sensor." `https://www.st.com/en/imaging-and-photonics-solutions/vl53l0x.html#documentation`.

[27] D. J. Rodríguez Patarroyo, R. A. Gómez Porras, and A. F. Campos Fajardo, "Sistemas de gestión de baterías (bms) y su importancia para los sistemas de almacenamiento de baterías (bess)," *Tecnura*, vol. 19, p. 51–56, nov. 2015.

[28] B. Emanet and Kıyak, "Performance effective battery management system(bms) design of mini electric vehicles," in *2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, pp. 744–749, 2021.

[29] "The python profilers." `https://docs.python.org/3/library/profile.html`.

[30] "timeit — measure execution time of small code snippets." https://docs.python.org/3/library/timeit.html.

[31] "Ads1x15 python library." https://github.com/chandrawi/ADS1x15-ADC/tree/main.

[32] "Ads1115 datasheet." https://www.ti.com/product/es-mx/ADS1115.

[33] "Ads1015 datasheet." https://www.ti.com/product/es-mx/ADS1015.

[34] "Mcp3208 datasheet." https://www.microchip.com/en-us/product/mcp3208.

[35] NXP Semiconductors, *I^2C-bus specification and user manual (UM10204, Rev. 7.0)*. NXP Semiconductors, 2021.

[36] "Ups module 3s - raspberry pi." https://www.waveshare.com/wiki/UPS_Module_3S.

[37] "Add-on board / hat design guide." https://github.com/raspberrypi/hats/blob/master/designguide.md.

[38] "Raspberry pi hat+ specification." https://datasheets.raspberrypi.com/hat/hat-plus-specification.pdf.