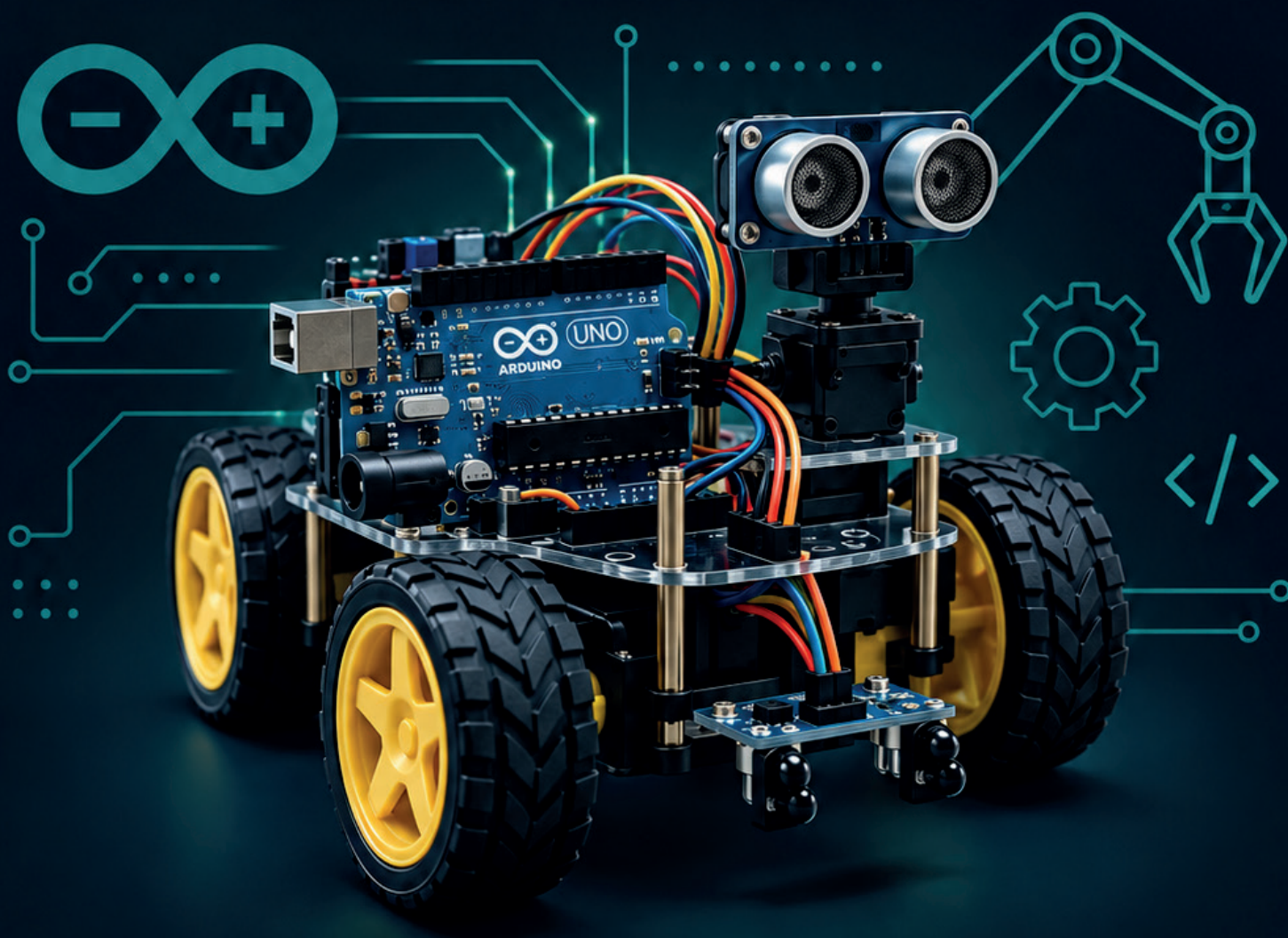


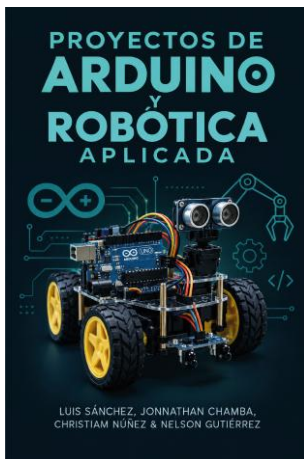
PROYECTOS DE **ARDUINO** Y **ROBÓTICA** APLICADA



LUIS SÁNCHEZ, JONNATHAN CHAMBA,
CHRISTIAM NÚÑEZ & NELSON GUTIÉRREZ

Proyectos de Arduino y Robótica Aplicada

**Luis Miguel Sánchez Muyulema
Jonnathan Ismael Chamba Cruz
Christiam Xavier Núñez Zavala
Nelson Ramiro Gutiérrez Suquillo**



Datos bibliográficos

ISBN:	978-9907-803-72-3
Título del libro:	Proyectos de Arduino y Robótica Aplicada
Autores:	Sánchez Muyulema, Luis Miguel Chamba Cruz, Jonnathan Ismael Núñez Zavala, Christiam Xavier Gutiérrez Suquillo, Nelson Ramiro
Editorial:	SAGA
Materia:	629.892 - Robots
Público objetivo:	Profesional / académico
Publicado:	2026-09-10
Número de edición:	1
Tamaño:	5Mb
Soporte:	Libro digital descargable
Formato:	Pdf (.pdf)
Idioma:	Español
DOI:	https://doi.org/10.63415/saga.2026.126

Prólogo

La robótica competitiva representa mucho más que la construcción de máquinas capaces de desplazarse, seguir una línea, detectar un oponente o ejecutar una tarea específica. Es un espacio donde convergen la ingeniería, la creatividad, la experimentación, la disciplina y el aprendizaje basado en problemas reales.

Este libro nace desde una visión aplicada de la enseñanza de la robótica. Su propósito es acompañar al lector en el diseño e implementación de sistemas embebidos mediante proyectos progresivos, conectados con situaciones reales de competencia. Cada capítulo propone una solución concreta, pero también busca desarrollar criterio técnico: saber por qué se elige un sensor, cómo se protege una batería, cómo se ajusta un controlador, cómo se analiza un error y cómo se documenta una mejora.

La obra está pensada para estudiantes, docentes, clubes de robótica y equipos que desean pasar de proyectos aislados a sistemas más integrados. A lo largo de los capítulos se utilizan plataformas accesibles como Arduino y ESP32, sin perder de vista conceptos avanzados como control PID, lógica difusa, odometría, telemetría, registro de datos, análisis con Python, TinyML y diagnóstico inteligente de fallas.

La experiencia en competencias demuestra que un robot ganador no depende únicamente de tener buenos componentes. Depende de la integración, la validación experimental, la documentación, la confiabilidad y la capacidad del equipo para aprender de cada prueba. Por ello, el libro no se limita a presentar códigos, sino que propone una metodología de trabajo basada en evidencia.

Esta obra reúne una propuesta práctica y progresiva para el aprendizaje de la robótica competitiva y los sistemas embebidos. A través de proyectos aplicados, se integran conceptos de electrónica, programación, control, sensores, telemetría, análisis de datos e inteligencia artificial, con el propósito de fortalecer la formación técnica, la experimentación sistemática y la resolución de problemas reales en contextos educativos y competitivos.

Presentación de la obra

La robótica competitiva se ha convertido en una herramienta educativa poderosa para la formación en ingeniería, tecnología y ciencias aplicadas. A diferencia de una práctica tradicional de laboratorio, una competencia exige que el sistema funcione en condiciones reales: con límites de tiempo, restricciones de peso, reglamentos, fallas imprevistas, variaciones de pista y presión operativa.

Este libro presenta una ruta progresiva para construir robots competitivos e inteligentes usando sistemas embebidos. Los primeros capítulos introducen la electrónica, la arquitectura de software y la seguridad básica. Posteriormente se desarrollan proyectos aplicados como Minisumo, telemetría de impactos, control de motores, sensores de borde, sistemas interactivos, wearables, medición de centro de masa, seguidores de línea y dashboards inalámbricos.

La segunda parte del libro avanza hacia sistemas móviles más complejos: visión artificial, robots exploradores, odometría, fusión sensorial y navegación por objetivos. Después se incorporan técnicas de control inteligente, como control difuso y control híbrido PID-Difuso, acompañadas por herramientas de cronometraje, data logger, análisis de datos y optimización experimental.

Finalmente, los últimos capítulos introducen aprendizaje automático básico, TinyML, diagnóstico inteligente de fallas e integración final. Esta estructura busca mostrar que la robótica competitiva moderna no solo consiste en programar movimientos, sino en diseñar sistemas capaces de medir, analizar, decidir, adaptarse y protegerse. Cada capítulo mantiene una organización similar: resultados de aprendizaje, planteamiento del problema, fundamentos teóricos, materiales, diseño electrónico, código, pruebas, errores frecuentes, actividades y referencias. De esta forma, el libro puede utilizarse tanto como material de clase como guía de desarrollo para clubes de robótica.

El lector no necesita dominar todos los temas desde el inicio. La propuesta es aprender construyendo. Cada proyecto puede implementarse de forma independiente o integrarse gradualmente en una plataforma más completa. El objetivo final es que el estudiante o equipo sea capaz de diseñar un robot competitivo documentado, validado y técnicamente defendible.

Acerca de los autores



Luis Miguel Sánchez Muyulema

Ingeniero en Electrónica Control y Redes Industriales

Magister en Sistemas de Control y Automatización Industrial

<https://orcid.org/0000-0001-5078-3734>

luis.sanchez@ute.edu.ec

Universidad UTE

Docente de la carrera de Mecatrónica, con una amplia experiencia en educación STEAM y robótica educativa. Ha liderado proyectos de investigación en automatización industrial y obtenido múltiples reconocimientos en competencias internacionales de robótica como: RobotChallenge China 2024, InkaChallenge IV Perú 2025. Su labor está enfocada en la investigación, desarrollo tecnológico y la promoción de la robótica.



Jonnathan Ismael Chamba Cruz

Ingeniero en Electrónica Control y Redes Industriales

Magister en Electrónica y Automatización

<https://orcid.org/0000-0003-2792-7271>

jonnathan.chamba@ute.edu.ec

Universidad UTE

Docente de la carrera de Ingeniería Electromecánica, con una amplia experiencia en el desarrollo de proyectos de investigación. Apasionado por la robótica, ha destacado por su participación en diversos concursos nacionales e internacionales, en los cuales ha obtenido reconocimientos por su excelencia (RobotChallenge China 2024). Su enfoque innovador y compromiso con la tecnología lo posicionan como un referente en su campo.



Christiam Xavier Núñez Zavala

Ingeniero en Electrónica, Control y Redes Industriales

Magíster en Sistemas de Control y Automatización Industrial.

<https://orcid.org/0000-0001-8162-5616>

cnunez@unach.edu.ec

Universidad Nacional de Chimborazo

Docente en la Facultad de Ciencias de la Educación Humanas y Tecnologías, Carrera de Pedagogía de la Informática. Miembro del grupo de Investigación Umayuk de la Universidad nacional de Chimborazo. Líneas de investigación de interés: electrónica analógica, electrónica de potencia, microcontroladores, visión artificial, sistemas de control clásicos y modernos.



Nelson Ramiro Gutiérrez Suquillo

Ingeniero Informático Mención Redes de Información

PhD en Automatización y Control de Procesos Tecnológicos y de Producción.

<https://orcid.org/0000-0003-2160-528X>

nelson.gutierrez@ute.edu.ec

Universidad UTE

Docente investigador en la carrera de Mecatrónica en la Facultad de Ciencias de la Ingeniería e Industrias. Líneas de investigación: mecatrónica, robótica y energías. Con experiencia en dirección de proyectos de automatización y comunicaciones industriales en diversas empresas de manufactura. Gestión de proyectos en infraestructura y gestión energética en plantas industriales.

Índice general

Prólogo	ii
Presentación de la obra	iv
Acerca de los autores	v
Capítulo 1. Introducción a la Robótica Competitiva	30
1.1. Introducción	30
1.2. Concepto de robótica competitiva	32
1.3. Beneficios educativos de la robótica competitiva.....	33
1.4. Relación entre la robótica competitiva y la metodología STEM.....	33
1.4.1. Ciencia	33
1.4.2. Tecnología	34
1.4.3. Ingeniería	34
1.4.4. Matemáticas.....	34
1.5. Categorías principales de competencia	35
1.5.1. Minisumo.....	35
1.5.2. Microsumo.....	36
1.5.3. Seguidores de línea.....	36
1.5.4. Robot Soccer	37
1.5.5. Robot Combate.....	37
1.5.6. Laberintos y exploración	37
1.6. Evolución tecnológica de los robots competitivos.....	37
1.6.1. Primera generación	37
1.6.2. Segunda generación.....	37
1.6.3. Tercera generación	37
1.6.4. Cuarta generación	38
1.7. Categorías principales de competencia	38
1.8. La robótica competitiva en Ecuador	39
1.9. Caso de estudio 1.1: Clubes universitarios de robótica	40
1.10. Arduino y ESP32 como plataformas de desarrollo.....	40

1.10.1. Arduino.....	40
1.10.2. ESP32	40
1.11. Organización general del libro.....	41
Actividades de aprendizaje	41
Preguntas de reflexión.....	42
Referencias.....	42
Capítulo 2. Electrónica para Robots Competitivos.....	43
2.1. Introducción	43
2.2. Arquitectura electrónica de un robot.....	44
2.3. Sistema de alimentación	45
2.3.1. Baterías LiPo	45
2.3.2. Baterías LiPo HV	46
2.3.3. Caso de estudio 2.1: batería de 8 celdas en un robot de sumo	46
2.4. Regulación y distribución de energía.....	47
2.4.1. Reguladores lineales	47
2.4.2. Convertidores DC – DC	47
2.4.3. Buenas prácticas de distribución	48
2.5. Microcontroladores	48
2.5.1. Arduino Nano	49
2.5.2. ESP32	49
2.5.3. Selección de plataforma	49
2.6. Sensores utilizados en robótica competitiva	50
2.6.1. Sensores infrarrojos reflectivos	50
2.6.2. Sensores de distancia	51
2.6.3. Preguntas de reflexión	51
2.6.4. Referencias del capítulo	51
2.7. Actuadores	52
2.7.1. Motores DC	52
2.7.2. Motores brushless	52
2.7.3. Servomotores	52

2.7.4. Motores paso a paso	52
2.7.5. Actuadores hápticos.....	53
2.8. Electrónica de potencia	53
2.8.1. Puentes H.....	53
2.8.2. ESC para motores brushless	54
2.9. Ruido eléctrico e interferencias	54
2.9.1. Técnicas de mitigación	55
2.10. Protección eléctrica y seguridad	55
2.10.1. Fusibles e interruptores.....	55
2.10.2. Protección contra inversión de polaridad	55
2.10.3. Protección contra sobretensiones.....	55
2.10.4. Supervisión de batería	55
2.11. Caso de estudio 2.2: electrónica de un robot Minisumo	56
2.12. Caso de estudio 2.3: electrónica de un robot de combate	57
2.13. Buenas prácticas de montaje.....	58
Actividades de aprendizaje	58
Preguntas de reflexión.....	58
Referencias.....	58
Capítulo 3. Arquitectura de Software para Sistemas Embebidos	60
3.1. Introducción	60
3.2. Sistemas embebidos en robótica	61
3.3. Estructura básica de un programa Arduino.....	62
3.3.1. Función setup.....	62
3.3.2. Función loop.....	63
3.4. Variables y tipos de datos	63
3.5. Programación bloqueante.....	64
3.6. Caso de estudio 3.1: retraso crítico en Minisumo	64
3.7. Programación no bloqueante con millis.....	64
3.8. Máquinas de estados	66
3.9. Funciones y modularidad.....	67

3.10. Interrupciones	68
3.11. PWM.....	69
3.12. Comunicación serial.....	70
3.13. Comunicación Bluetooth y aplicaciones móviles	71
3.14. Comunicación WiFi con ESP32	71
3.15. Almacenamiento no volátil	72
3.16. Filtros digitales.....	72
3.16.1. Promedio móvil	73
3.16.2. Promedio móvil	73
3.17. Introducción al control PID	73
3.18. Caso de estudio 3.2: seguidor de línea de alta velocidad.....	74
3.19. Buenas prácticas de programación embebida	75
3.20. Caso de estudio 3.3: arquitectura de software para Minisumo	75
Actividades de aprendizaje	76
Preguntas de reflexión.....	76
Referencias.....	76
Capítulo 4. Supervisor Inteligente de Seguridad para Robots Competitivos.....	77
4.1. Introducción	77
4.2. Planteamiento del problema.....	78
4.3. Descripción general del proyecto.....	79
4.4. Aplicación STEM	79
4.5. Objetivos del proyecto	80
4.5.1. Objetivo general	80
4.5.2. Objetivos específicos.....	80
4.6. Materiales y herramientas	80
4.7. Fundamento teórico	80
4.7.1. Medición de voltaje con divisor resistivo.....	80
4.7.2. Medición de corriente con ACS712	81
4.7.3. Detección de señal RC.....	82
4.8. Diseño electrónico	83

4.9. Organización modular del proyecto.....	84
4.10. Lógica de funcionamiento.....	84
4.11. Código del proyecto.....	85
4.12. Pruebas experimentales.....	90
4.12.1. Prueba 1: medición de voltaje	90
4.12.2. Prueba 2: medición de corriente	90
4.12.3. Prueba 3: pérdida de señal RC.....	91
4.12.4. Prueba 4: activación de salida segura	91
4.13. Extensión con pantalla OLED	91
4.14. Extensión con App Inventor	92
4.15. Análisis de resultados	92
4.16. Errores frecuentes	93
4.17. Proyecto de extensión	94
Actividades de aprendizaje	94
Preguntas de reflexión.....	94
Referencias.....	94
Capítulo 5. Telemetría de Impactos con MPU6050 y EEPROM	96
5.1. Introducción	96
5.2. Planteamiento del problema.....	97
5.3. Descripción general del proyecto.....	98
5.4. Aplicación STEM	98
5.5. Objetivos del proyecto	99
5.5.1. Objetivo general	99
5.5.2. Objetivos específicos.....	99
5.6. Materiales y herramientas	99
5.7. Fundamento teórico	100
5.7.1. Sensor MPU6050.....	100
5.7.2. Comunicación I2C.....	100
5.7.3. Aceleración en tres ejes	101
5.7.4. Detección de impacto	101

5.7.5. Registro en EEPROM.....	101
5.8. Diseño electrónico	102
5.9. Organización modular del programa	103
5.10. Código del proyecto.....	103
5.11. Calibración del sistema	108
5.11.1. Prueba en reposo.....	108
5.11.2. Prueba de orientación	108
5.11.3. Prueba de impacto suave	108
5.11.4. Prueba de umbral.....	108
5.12. Pruebas experimentales en robots competitivos	109
5.12.1. Minisumo.....	109
5.12.2. Robot de combate	109
5.12.3. Seguidor de línea	109
5.13. Análisis de datos	110
5.14. Visualización en MATLAB.....	110
5.15. Errores frecuentes	111
5.16. Proyecto de extensión	111
Actividades de aprendizaje	112
Preguntas de reflexión.....	112
Referencias.....	112
Capítulo 6. Control Seguro de Motores Brushless con ESC	113
6.1. Introducción	113
6.2. Planteamiento del problema.....	114
6.3. Descripción general del proyecto.....	114
6.4. Aplicación STEM	115
6.5. Objetivos del proyecto	116
6.5.1. Objetivo general	116
6.5.2. Objetivos específicos.....	116
6.6. Materiales y herramientas	116
6.7. Consideraciones de seguridad.....	117

6.8. Fundamento teórico	117
6.8.1. Motor brushless	117
6.8.2. Diseño e implementación	118
6.8.3. Rampa de aceleración	118
6.8.4. Estados de operación	118
6.9. Diseño electrónico	119
6.10. Código del proyecto	120
6.11. Procedimiento de prueba segura	124
6.11.1. Prueba 1: verificación sin batería del ESC	124
6.11.2. Prueba 2: señal de control	124
6.11.3. Prueba 3: montaje mecánico	124
6.11.4. Prueba 4: armado	124
6.11.5. Prueba 5: rampa progresiva	124
6.11.6. Prueba 6: emergencia	125
6.12. Calibración del ESC	125
6.13. Análisis de resultados	125
6.14. Errores frecuentes	126
6.15. Extensión del proyecto	126
Actividades de aprendizaje	127
Preguntas de reflexión	127
Referencias	127
Capítulo 7. Sensores de Borde Inteligentes con Filtrado Temporal	129
7.1. Introducción	129
7.2. Planteamiento del problema	130
7.3. Descripción general del proyecto	131
7.4. Aplicación STEM	132
7.5. Objetivos del proyecto	132
7.5.1. Objetivo general	132
7.5.2. Objetivos específicos	132
7.6. Materiales y herramientas	132

7.7. Fundamento teórico	133
7.7.1. Sensores reflectivos	133
7.7.2. Conversión analógico-digital.....	134
7.7.3. Umbral de detección	134
7.7.4. Histéresis	134
7.7.5. Confirmación temporal.....	135
7.7.6. Filtro exponencial	135
7.8. Diseño electrónico	136
7.9. Calibración inicial	137
7.10. Organización modular del programa	138
7.11. Código del proyecto.....	139
7.12. Integración con un robot Minisumo.....	143
7.13. Integración con un seguidor de línea	144
7.14. Pruebas experimentales.....	145
7.14.1. Prueba 1: lectura en reposo.....	145
7.14.2. Prueba 2: detección de borde blanco	145
7.15. Ajuste de parámetros.....	146
7.15.1. Umbral alto	146
7.15.2. Umbral bajo	146
7.15.3. Factor de filtro	146
7.15.4. Tiempo de confirmación.....	146
7.16. Errores frecuentes	147
7.17. Proyecto de extensión	147
Actividades de aprendizaje	148
Preguntas de reflexión.....	148
Referencias.....	148
Capítulo 8. Cinturón Interactivo RFID con Efectos de Luz y Sonido	149
8.1. Introducción	149
8.2. Planteamiento del problema.....	150
8.3. Descripción general del proyecto.....	150

8.4. Aplicación STEM	151
8.5. Objetivos del proyecto	151
8.5.1. Objetivo general	151
8.5.2. Objetivos específicos.....	152
8.6. Materiales y herramientas	152
8.7. Fundamento teórico	153
8.7.1. Tecnología RFID	153
8.7.2. Comunicación SPI	153
8.7.3. LEDs direccionables WS2812B	153
8.7.4. Módulo DFPlayer Mini	153
8.8. Diseño electrónico	154
8.9. Diseño de interacción.....	155
8.10. Preparación de la memoria microSD	156
8.11. Código del proyecto.....	157
8.12. Lectura de UID de tarjetas	163
8.13. Diseño mecánico y fabricación.....	164
8.14. Pruebas experimentales.....	164
8.14.1. Prueba 1: lector RFID.....	164
8.14.2. Prueba 2: LEDs WS2812B.....	164
8.14.3. Prueba 3: DFPlayer Mini.....	165
8.14.4. Prueba 4: integración completa	165
8.15. Errores frecuentes	165
8.16. Proyecto de extensión	166
Actividades de aprendizaje	166
Preguntas de reflexión.....	167
Referencias.....	167
Capítulo 9. Guantelete de Control por Gestos con ESP32.....	168
9.1. Introducción	168
9.2. Planteamiento del problema.....	169
9.3. Descripción general del proyecto.....	169

9.4. Aplicación STEM	170
9.5. Objetivos del proyecto	170
9.5.1. Objetivo general	170
9.5.2. Objetivos específicos.....	171
9.6. Materiales y herramientas	171
9.7. Fundamento teórico	172
9.7.1. Sensores flexibles	172
9.7.2. MPU6050	172
9.7.3. Bluetooth clásico en ESP32.....	173
9.8. Diseño electrónico	173
9.9. Diseño de gestos	174
9.10. Calibración de sensores flexibles.....	175
9.11. Código del proyecto.....	176
9.12. Protocolo de comandos	181
9.13. Receptor básico para robot.....	181
9.14. Integración con App Inventor	183
9.15. Diseño mecánico y ergonomía.....	184
9.16. Pruebas experimentales.....	184
9.16.1. Prueba 1: sensores flexibles.....	184
9.16.2. Prueba 2: MPU6050	184
9.16.3. Prueba 3: Bluetooth	184
9.16.4. Prueba 4: integración con robot.....	185
9.17. Errores frecuentes	185
9.18. Proyecto de extensión	186
Actividades de aprendizaje	186
Preguntas de reflexión.....	186
Referencias.....	187
Capítulo 10. Medición del Centro de Masa con Celdas de Carga y HX711	188
10.1. Introducción	188
10.2. Planteamiento del problema.....	189

10.3. Descripción general del proyecto.....	189
10.4. Aplicación STEM	190
10.5. Objetivos del proyecto	190
10.5.1. Objetivo general	190
10.5.2. Objetivos específicos.....	190
10.6. Materiales y herramientas	191
10.7. Fundamento teórico	192
10.7.1. Centro de masa	192
10.7.2. Sumatoria de fuerzas	192
10.7.3. Cálculo de coordenadas del centro de masa	192
10.7.4. Celda de carga	193
10.7.5. Módulo HX711.....	193
10.8. Diseño electrónico	193
10.9. Diseño mecánico de la plataforma.....	195
10.10. Calibración de celdas de carga.....	195
10.11. Código de calibración	196
10.12. Código del proyecto.....	197
10.13. Interpretación de resultados	200
10.14. Aplicación en Minisumo.....	201
10.15. Aplicación en seguidores de línea.....	201
10.16. Pruebas experimentales.....	202
10.16.1. Prueba 1: peso total	202
10.16.2. Prueba 2: desplazamiento lateral	202
10.16.3. Prueba 3: desplazamiento longitudinal.....	202
10.16.4. Prueba 4: robot real	202
10.17. Análisis de datos	203
10.18. Errores frecuentes	204
10.19. Proyecto de extensión	204
Actividades de aprendizaje	205
Preguntas de reflexión.....	205

Referencias.....	205
Capítulo 11. Seguidor de Línea Competitivo con PID y Barra de 16 Sensores	206
11.1. Introducción	206
11.2. Planteamiento del problema.....	207
11.3. Descripción general del proyecto.....	207
11.4. Aplicación STEM	208
11.5. Objetivos del proyecto	208
11.5.1. Objetivo general	208
11.5.2. Objetivos específicos.....	209
11.6. Materiales y herramientas	209
11.7. Fundamento teórico	210
11.7.1. Sensores reflectivos para seguimiento de línea	210
11.7.2. Multiplexación de sensores	210
11.7.3. Normalización de lecturas	211
11.7.4. Cálculo de posición de línea.....	211
11.7.5. Control PID	211
11.8. Diseño electrónico	212
11.9. Calibración de la barra de sensores.....	213
11.10. Código del proyecto.....	215
11.11. Ajuste experimental del PID.....	220
11.11.1. Ajuste de K_p	220
11.11.2. Ajuste de K_d	220
11.11.3. Ajuste de K_i	220
11.12. Frenado en curvas cerradas	220
11.13. Relación entre sensores y mecánica.....	221
11.14. Pruebas experimentales.....	222
11.14.1. Prueba 1: lectura de sensores.....	222
11.14.2. Prueba 2: posición de línea.....	222
11.14.3. Prueba 3: seguimiento lento	222
11.14.4. Prueba 4: aumento progresivo de velocidad.....	222

11.14.5. Prueba 5: curvas cerradas	222
11.15. Errores frecuentes	223
11.16. Proyecto de extensión	223
Actividades de aprendizaje	224
Preguntas de reflexión.....	224
Referencias.....	224
Capítulo 12. PID Web Tuning con ESP32.....	225
12.1. Introducción	225
12.2. Planteamiento del problema.....	226
12.3. Descripción general del proyecto.....	226
12.4. Aplicación STEM	227
12.5. Objetivos del proyecto	227
12.5.1. Objetivo general	227
12.5.2. Objetivos específicos.....	228
12.6. Materiales y herramientas	228
12.7. Fundamento teórico	228
12.7.1. ESP32 como punto de acceso.....	228
12.7.2. Servidor web embebido	229
12.7.3. Memoria no volátil con Preferences.....	229
12.7.4. Parámetros PID ajustables	229
12.8. Diseño electrónico	229
12.9. Diseño de la interfaz web.....	231
12.10. Código del proyecto.....	231
12.11. Integración con el seguidor de línea	238
12.12. Procedimiento de uso.....	239
12.13. Tabla de pruebas	239
12.14. Criterios de ajuste	239
12.14.1. Ajuste inicial.....	239
12.14.2. Control de oscilación	239
12.14.3. Velocidad base.....	239

12.14.4. Uso del término integral	240
12.15. Seguridad durante pruebas	240
12.16. Errores frecuentes	240
12.17. Proyecto de extensión	241
Actividades de aprendizaje	241
Preguntas de reflexión.....	242
Referencias.....	242
Capítulo 13. Visión Artificial con ESP32-CAM y OpenCV	243
13.1. Introducción	243
13.2. Planteamiento del problema.....	244
13.3. Descripción general del proyecto.....	244
13.4. Aplicación STEM	245
13.5. Objetivos del proyecto	245
13.5.1. Objetivo general	245
13.5.2. Objetivos específicos.....	245
13.6. Materiales y herramientas	246
13.7. Fundamento teórico	246
13.7.1. ESP32-CAM.....	246
13.7.2. OpenCV	247
13.7.3. Espacios de color	247
13.7.4. Segmentación por umbral.....	247
13.7.5. Contornos y centroide.....	247
13.8. Diseño electrónico	248
13.9. Código para ESP32-CAM.....	249
13.10. Prueba del video.....	251
13.11. Código Python con OpenCV	252
13.12. Interpretación de la salida	254
13.13. Ajuste de color HSV	255
13.14. Integración con un robot móvil.....	255
13.15. Limitaciones prácticas	256

13.16. Pruebas experimentales.....	256
13.16.1. Prueba 1: transmisión de cámara.....	256
13.16.2. Prueba 2: captura desde Python.....	256
13.16.3. Prueba 3: detección de color.....	257
13.16.4. Prueba 4: posición del objeto	257
13.16.5. Prueba 5: integración con robot.....	257
13.17. Errores frecuentes	257
13.18. Proyecto de extensión	257
Actividades de aprendizaje	258
Preguntas de reflexión.....	258
Referencias.....	258
Capítulo 14. Robot Explorador WiFi con ESP32-CAM.....	260
14.1. Introducción	260
14.2. Planteamiento del problema.....	261
14.3. Descripción general del proyecto.....	261
14.4. Aplicación STEM	262
14.5. Objetivos del proyecto	262
14.5.1. Objetivo general	262
14.5.2. Objetivos específicos.....	262
14.6. Materiales y herramientas	263
14.7. Fundamento teórico	263
14.7.1. Control remoto por WiFi	263
14.7.2. Arquitectura con dos microcontroladores.....	264
14.7.3. Comunicación serial entre módulos	264
14.7.4. Control diferencial.....	264
14.8. Diseño electrónico	264
14.9. Diseño de la interfaz web.....	266
14.10. Código para Arduino Nano: control de motores.....	267
14.11. Código para ESP32-CAM: interfaz de comandos	270
14.12. Integración del video.....	273

14.13. Pruebas experimentales.....	273
14.13.1. Prueba 1: control de motores sin cámara.....	273
14.13.2. Prueba 2: video estable.....	273
14.13.3. Prueba 3: interfaz web.....	274
14.13.4. Prueba 4: seguridad por pérdida de comandos.....	274
14.13.5. Prueba 5: prueba de exploración.....	274
14.14. Consideraciones de alimentación.....	274
14.15. Problemas frecuentes.....	275
14.16. Proyecto de extensión.....	275
Actividades de aprendizaje.....	276
Preguntas de reflexión.....	276
Referencias.....	276
Capítulo 15. Odometría y Control de Velocidad con Encoders.....	277
15.1. Introducción.....	277
15.2. Planteamiento del problema.....	278
15.3. Descripción general del proyecto.....	278
15.4. Aplicación STEM.....	279
15.5. Objetivos del proyecto.....	279
15.5.1. Objetivo general.....	279
15.5.2. Objetivos específicos.....	279
15.6. Materiales y herramientas.....	280
15.7. Fundamento teórico.....	280
15.7.1. Encoder incremental.....	280
15.7.2. Distancia recorrida.....	281
15.7.3. Velocidad y RPM.....	281
15.7.4. Odometría diferencial.....	281
15.7.5. Error acumulado.....	282
15.8. Diseño electrónico.....	282
15.9. Calibración inicial.....	284
15.10. Código del proyecto.....	284

15.11. Validación experimental	291
15.11.1. Prueba 1: conteo de pulsos	291
15.11.2. Prueba 2: una revolución	291
15.11.3. Prueba 3: distancia recta	292
15.11.4. Prueba 4: giro sobre su eje.....	292
15.11.5. Prueba 5: control de velocidad	292
15.12. Ajuste del control de velocidad.....	292
15.13. Integración con robot explorador autónomo	293
15.14. Limitaciones de la odometría.....	294
15.15. Errores frecuentes	294
15.16. Proyecto de extensión	294
Actividades de aprendizaje	295
Preguntas de reflexión.....	295
Referencias.....	295
Capítulo 16. Navegación por Objetivos con Odometría e IMU	297
16.1. Introducción	297
16.2. Planteamiento del problema.....	298
16.3. Descripción general del proyecto.....	299
16.4. Aplicación STEM	299
16.5. Objetivos del proyecto	299
16.5.1. Objetivo general	299
16.5.2. Objetivos específicos.....	300
16.6. Materiales y herramientas	300
16.7. Fundamento teórico	301
16.7.1. Distancia hacia un objetivo	301
16.7.2. Ángulo hacia un objetivo.....	301
16.7.3. Error angular.....	301
16.7.4. Tolerancias	301
16.7.1. Estrategia de navegación por punto [3]	302
16.8. Diseño electrónico	302

16.9. Diseño de la máquina de estados	303
16.10. Código del proyecto	304
16.11. Trayectorias básicas	315
16.11.1. Trayectoria en línea recta	315
16.11.2. Trayectoria cuadrada	315
16.11.3. Trayectoria triangular	315
16.12. Validación experimental	316
16.12.1. Prueba 1: avance a un punto	316
16.12.2. Prueba 2: giro hacia objetivo lateral	316
16.12.3. Prueba 3: trayectoria cuadrada	316
16.12.4. Prueba 4: repetibilidad.....	316
16.12.5. Prueba 5: superficie diferente	316
16.13. Análisis del error	317
16.14. Mejoras recomendadas.....	317
16.15. Integración con el dashboard de telemetría	317
16.16. Limitaciones del sistema	318
16.17. Errores frecuentes	318
16.18. Proyecto de extensión	319
Actividades de aprendizaje	319
Preguntas de reflexión.....	320
Referencias.....	320
Capítulo 17. Control Difuso para Robots Competitivos	321
17.1. Introducción	321
17.2. Planteamiento del problema.....	322
17.3. Descripción general del proyecto.....	322
17.4. Aplicación STEM	323
17.5. Objetivos del proyecto	323
17.5.1. Objetivo general	323
17.5.2. Objetivos específicos.....	324
17.6. Materiales y herramientas	324

17.7. Fundamento teórico	325
17.7.1. Lógica clásica y lógica difusa.....	325
17.7.2. Variables lingüísticas.....	325
17.7.3. Función de pertenencia triangular	325
17.7.4. Reglas difusas	325
17.8. Variables del controlador.....	326
17.9. Base de reglas	326
17.10. Diseño electrónico	327
17.11. Código del proyecto.....	327
17.12. Comparación con máquina de estados.....	332
17.13. Procedimiento de prueba.....	333
17.14. Ajuste del controlador difuso.....	333
17.15. Errores frecuentes	334
17.16. Proyecto de extensión	334
Actividades de aprendizaje	334
Preguntas de reflexión.....	335
Referencias.....	335
Capítulo 18. Registro de Datos con microSD para Robots Competitivos	336
18.1. Introducción	336
18.2. Planteamiento del problema.....	337
18.3. Descripción general del proyecto.....	338
18.4. Aplicación STEM	338
18.5. Objetivos del proyecto	339
18.5.1. Objetivo general	339
18.5.2. Objetivos específicos.....	339
18.6. Materiales y herramientas	339
18.7. Fundamento teórico	340
18.7.1. Comunicación SPI.....	340
18.7.2. Formato CSV	340
18.7.3. Frecuencia de muestreo	340

18.7.4. Registro no bloqueante	341
18.7.5. Medición de voltaje	341
18.8. Diseño electrónico	341
18.9. Selección de variables	342
18.10. Código del proyecto	343
18.11. Integración con un seguidor de línea	348
18.12. Integración con un Minisumo	349
18.13. Análisis en hoja de cálculo	349
18.14. Frecuencia de registro recomendada	350
18.15. Buenas prácticas.....	350
18.16. Errores frecuentes	350
18.17. Proyecto de extensión	351
Actividades de aprendizaje	351
Preguntas de reflexión.....	351
Referencias.....	352
Capítulo 19. Análisis de Datos y Optimización Experimental con Python	353
19.1. Introducción	353
19.2. Planteamiento del problema.....	354
19.3. Descripción general del proyecto.....	354
19.4. Aplicación STEM	355
19.5. Objetivos del proyecto	355
19.5.1. Objetivo general	355
19.5.2. Objetivos específicos.....	355
19.6. Materiales y herramientas	356
19.7. Estructura recomendada de archivos	356
19.8. Lectura de un archivo CSV	357
19.9. Revisión inicial de datos	357
19.10. Métricas de desempeño.....	358
19.10.1. Error máximo y error promedio.....	358
19.10.2. Saturación de PWM.....	358

19.10.3. Caída de voltaje	359
19.10.4. Eventos de falla	359
19.11. Gráficas básicas	359
19.11.1. Error de línea en el tiempo	359
19.11.2. PWM de motores	360
19.11.3. Voltaje de batería.....	360
19.12. Comparación de dos perfiles.....	360
19.13. Comparación de varios perfiles	361
19.14. Tabla de decisión	362
19.15. Análisis específico para seguidores de línea.....	362
19.16. Análisis específico para Minisumo	362
19.17. Generación de un reporte simple	363
19.18. Metodología de optimización experimental.....	363
19.19. Ejemplo de interpretación	364
19.20. Errores frecuentes	364
19.21. Proyecto de extensión	365
Actividades de aprendizaje	365
Preguntas de reflexión.....	365
Referencias.....	366
Capítulo 20. Aprendizaje Automático Básico para Robots Competitivos.....	367
20.1. Introducción	367
20.2. Planteamiento del problema.....	368
20.3. Descripción general del proyecto.....	369
20.4. Aplicación STEM	369
20.5. Objetivos del proyecto	369
20.5.1. Objetivo general	369
20.5.2. Objetivos específicos.....	370
20.6. Materiales y herramientas	370
20.7. Fundamento teórico	371
20.7.1. Características.....	371

20.7.2. Etiqueta.....	371
20.7.3. Clasificador.....	371
20.7.4. Árbol de decisión.....	371
20.7.5. Matriz de confusión	372
20.8. Construcción del conjunto de datos	372
20.9. Etiquetado de datos	373
20.10. Etiquetado inicial con Python	373
20.11. Entrenamiento de un árbol de decisión	374
20.12. Visualización del árbol	374
20.13. Interpretación de reglas	375
20.14. Implementación en Arduino	375
20.15. Uso del clasificador en un seguidor de línea	377
20.16. Uso del clasificador en Minisumo	378
20.17. Evaluación del modelo.....	379
20.18. Validación en el robot real.....	380
20.19. Buenas prácticas.....	381
20.20. Limitaciones.....	381
20.21. Proyecto de extensión	381
Actividades de aprendizaje	382
Preguntas de reflexión.....	382
Referencias.....	383
Capítulo 21. Integración Final y Preparación de un Robot Competitivo Inteligente.....	384
21.1. Introducción	384
21.2. Planteamiento del problema.....	385
21.3. Descripción general del proyecto.....	386
21.4. Aplicación STEM	386
21.5. Objetivos del proyecto	387
21.5.1. Objetivo general	387
21.5.2. Objetivos específicos.....	387
21.6. Arquitectura modular de hardware	387

21.7. Arquitectura modular de software	388
21.8. Prioridad de seguridad	389
21.9. Estructura base de código	389
21.10. Perfiles de competencia	393
21.11. Bitácora técnica.....	393
21.12. Control de versiones	394
21.13. Validación mecánica.....	395
21.14. Validación eléctrica	395
21.15. Validación de software	396
21.16. Pruebas finales por categoría	396
21.16.1. Seguidor de línea	396
21.16.2. Minisumo.....	396
21.16.3. Robot explorador	396
21.17. Plan de competencia	397
21.18. Gestión de riesgos	398
21.19. Documentación posterior a la competencia	398
21.20. Integración con investigación y docencia	399
21.21. Buenas prácticas.....	399
21.22. Errores frecuentes	399
21.23. Proyecto integrador final.....	400
Actividades de aprendizaje	401
Preguntas de reflexión.....	401
Referencias.....	401
Glosario.....	402

Capítulo 1. Introducción a la Robótica Competitiva

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Explicar el concepto de robótica competitiva y su relación con la formación en áreas STEM.
- Identificar las principales categorías de competencia robótica y sus desafíos técnicos.
- Reconocer la importancia de las plataformas de hardware abierto en la democratización de la robótica.
- Analizar el papel de los clubes universitarios de robótica como espacios de aprendizaje, innovación y vinculación.
- Comprender la estructura general del libro y la función de Arduino y ESP32 en los proyectos desarrollados.

1.1. Introducción

La robótica constituye una de las áreas tecnológicas de mayor crecimiento e impacto en la actualidad. Su influencia se extiende desde aplicaciones industriales y médicas hasta sistemas de exploración espacial, agricultura de precisión, logística automatizada, vehículos autónomos, dispositivos inteligentes y plataformas de asistencia personal. Este desarrollo ha sido impulsado por la convergencia de múltiples disciplinas, entre ellas la electrónica, la informática, la mecánica, las telecomunicaciones, la inteligencia artificial y el control automático [1].

Dentro de este amplio campo surge la robótica competitiva, una modalidad orientada al diseño, construcción y programación de robots capaces de resolver desafíos específicos bajo condiciones reglamentadas. A diferencia de los proyectos académicos tradicionales, cuyo objetivo suele ser demostrar el funcionamiento básico de un sistema, las competencias de robótica exigen niveles elevados de desempeño, robustez, eficiencia y confiabilidad. Un robot competitivo no solo debe funcionar correctamente; debe hacerlo en el menor tiempo posible, resistir condiciones exigentes y responder adecuadamente ante situaciones imprevistas.

La robótica competitiva se ha convertido en una herramienta educativa de gran valor debido a que permite integrar conocimientos teóricos y prácticos en entornos altamente motivadores. Los estudiantes no solo aprenden conceptos relacionados con sensores, actuadores, programación o diseño mecánico, sino que también desarrollan competencias asociadas al trabajo en equipo, la gestión de proyectos, la resolución de problemas, la toma de decisiones bajo presión y la innovación tecnológica [2].

Durante las últimas décadas, las competencias de robótica han evolucionado significativamente. Los primeros robots participantes eran sistemas relativamente simples, contruidos a partir de

circuitos discretos, sensores básicos y algoritmos elementales. Actualmente, es común encontrar plataformas que incorporan sistemas embebidos avanzados, comunicaciones inalámbricas, visión artificial, control PID, telemetría, análisis de datos y estrategias de comportamiento adaptativo.

La disponibilidad de plataformas de desarrollo como Arduino y ESP32 ha democratizado el acceso a estas tecnologías, permitiendo que estudiantes, docentes y entusiastas desarrollen soluciones cada vez más complejas con costos relativamente reducidos. Arduino se ha consolidado como una plataforma de prototipado electrónico de código abierto ampliamente utilizada en educación y desarrollo de proyectos interactivos [3]. Por su parte, el ESP32 integra capacidades de comunicación WiFi y Bluetooth, lo que lo convierte en una alternativa especialmente atractiva para aplicaciones de telemetría, Internet de las Cosas y control remoto [4].

En Ecuador, la robótica competitiva ha experimentado un crecimiento sostenido gracias al trabajo realizado por instituciones educativas, universidades, clubes de robótica y comunidades tecnológicas que promueven la participación estudiantil en eventos nacionales e internacionales. Esta evolución ha permitido que equipos ecuatorianos desarrollen proyectos cada vez más sofisticados y obtengan resultados destacados en categorías como Minisumo, Microsumo, Robot Soccer, seguidores de línea y robots de combate.

Este libro surge de la experiencia acumulada en docencia universitaria, desarrollo de proyectos mecatrónicos, formación de estudiantes y participación en competencias de robótica. Su propósito es proporcionar al lector una guía práctica que combine robótica competitiva, sistemas embebidos, instrumentación electrónica, comunicaciones, control y aplicaciones móviles mediante el desarrollo de proyectos aplicados basados en Arduino y ESP32.

Más que una colección de ejercicios aislados, la obra propone una metodología estructurada de desarrollo que integra diseño electrónico, programación, control, pruebas experimentales y documentación técnica. Cada capítulo está organizado para que el lector avance de manera progresiva desde los fundamentos hasta la implementación de sistemas robóticos más completos, favoreciendo la reutilización de soluciones, el análisis de resultados y la mejora continua de los proyectos.



Figura 1.1: Participación de estudiantes y docentes en competencias de robótica.

1.2. Concepto de robótica competitiva

La robótica competitiva puede definirse como la disciplina orientada al diseño, construcción y programación de robots capaces de ejecutar tareas específicas dentro de un entorno reglamentado, con el objetivo de superar a otros participantes o alcanzar el mejor desempeño posible.

A diferencia de otras aplicaciones robóticas donde el entorno de operación suele ser relativamente predecible, las competencias presentan escenarios dinámicos que exigen tiempos de respuesta reducidos y un elevado grado de confiabilidad. Los robots deben ser capaces de interpretar información proveniente de sensores, tomar decisiones en tiempo real y ejecutar acciones precisas bajo condiciones de incertidumbre.

Dependiendo de la categoría, los objetivos pueden incluir la búsqueda y expulsión de oponentes, el seguimiento de trayectorias complejas, el transporte de objetos, la navegación autónoma, la cooperación entre múltiples robots o incluso el combate directo mediante mecanismos especialmente diseñados para este propósito.

La robótica competitiva constituye un excelente laboratorio de aprendizaje porque integra simultáneamente conocimientos de mecánica, electrónica, programación, control automático, procesamiento de señales y comunicaciones. Esta característica la convierte en una herramienta ideal para la formación de futuros profesionales en áreas STEM.

La robótica competitiva no debe entenderse únicamente como una actividad extracurricular. Cuando se diseña de manera adecuada, se convierte en una estrategia de aprendizaje activo que permite aplicar conocimientos de ingeniería en problemas reales, medibles y retadores.

1.3. Beneficios educativos de la robótica competitiva

La participación en competencias de robótica genera beneficios que trascienden el ámbito estrictamente técnico. Desde la perspectiva académica, permite reforzar conceptos estudiados en asignaturas relacionadas con electrónica, programación, sistemas embebidos, control automático, diseño mecánico, instrumentación y comunicaciones.

Los estudiantes tienen la oportunidad de aplicar conocimientos teóricos en situaciones reales, identificando limitaciones prácticas que suelen estar ausentes en los ejercicios tradicionales de aula. Por ejemplo, un circuito que funciona correctamente en una protoboard puede fallar dentro de un robot debido a vibraciones, interferencias electromagnéticas o caídas de voltaje. Del mismo modo, un algoritmo probado en simulación puede presentar un comportamiento distinto al interactuar con sensores reales afectados por ruido, reflexión variable o condiciones ambientales cambiantes.

En el ámbito profesional, las competencias fomentan habilidades asociadas al liderazgo, la comunicación efectiva, la gestión de recursos, la planificación de actividades y la toma de decisiones bajo presión. Estas competencias son altamente valoradas en entornos industriales, académicos y de investigación.

Asimismo, la naturaleza multidisciplinaria de la robótica promueve el trabajo colaborativo entre estudiantes de diferentes áreas de formación, favoreciendo el desarrollo de proyectos integradores y fortaleciendo la cultura de innovación tecnológica. Un robot competitivo requiere normalmente la participación de perfiles complementarios: diseño mecánico, electrónica, programación, manufactura, documentación, pruebas y estrategia.

Finalmente, la participación en eventos nacionales e internacionales permite establecer redes de colaboración, intercambiar experiencias, conocer tendencias tecnológicas emergentes y comparar el nivel de desarrollo propio con el de otros equipos.

1.4. Relación entre la robótica competitiva y la metodología STEM

La metodología STEM (Science, Technology, Engineering and Mathematics) constituye uno de los enfoques educativos más influyentes de las últimas décadas. Su objetivo principal es integrar conocimientos científicos, tecnológicos, de ingeniería y matemáticos mediante experiencias de aprendizaje centradas en la resolución de problemas reales. La robótica competitiva representa una de las aplicaciones más completas de esta metodología debido a que involucra simultáneamente las cuatro áreas fundamentales del enfoque STEM.

1.4.1. Ciencia

La construcción y programación de robots requiere comprender diversos fenómenos físicos que influyen directamente en el comportamiento del sistema. Entre los conceptos científicos más relevantes se encuentran el movimiento, la dinámica, la fricción, la transferencia de energía, el

magnetismo, la propagación de ondas electromagnéticas y el comportamiento térmico de los componentes electrónicos.

Por ejemplo, en un robot Minisumo la capacidad de empujar a un adversario depende en gran medida del coeficiente de fricción existente entre las ruedas y la superficie del dohyo. Del mismo modo, la estabilidad de un robot móvil está condicionada por la distribución de masas y la posición de su centro de gravedad.

1.4.2. Tecnología

La tecnología proporciona las herramientas necesarias para implementar soluciones funcionales. En robótica competitiva es habitual utilizar microcontroladores, sensores digitales y analógicos, sistemas de comunicación inalámbrica, software de diseño asistido por computadora, herramientas de simulación y plataformas de fabricación digital.

Actualmente, plataformas como Arduino y ESP32 permiten desarrollar sistemas complejos con una inversión relativamente reducida, facilitando el acceso de estudiantes y aficionados al diseño de robots avanzados [1, 2].

1.4.3. Ingeniería

La ingeniería constituye el núcleo del proceso de diseño. Los participantes deben analizar problemas, proponer soluciones, construir prototipos, realizar pruebas y optimizar continuamente sus diseños. Las actividades típicas incluyen selección de materiales, diseño de chasis, integración electrónica, desarrollo de algoritmos, optimización energética, manufactura y validación experimental.

El proceso de ingeniería rara vez conduce a una solución perfecta desde el primer intento. Por el contrario, requiere múltiples ciclos de diseño, construcción, prueba y mejora. Esta dinámica convierte a la robótica competitiva en un entorno ideal para el aprendizaje basado en proyectos.

1.4.4. Matemáticas

Las matemáticas proporcionan las herramientas necesarias para modelar y analizar el comportamiento de los sistemas robóticos. Entre sus aplicaciones más frecuentes se encuentran la geometría, la trigonometría, el álgebra, la estadística, el cálculo diferencial y el control automático.

Por ejemplo, el ajuste de un controlador PID, el cálculo del centro de masa, la estimación de distancias mediante sensores o el análisis de datos de telemetría requieren una sólida comprensión de conceptos matemáticos fundamentales.

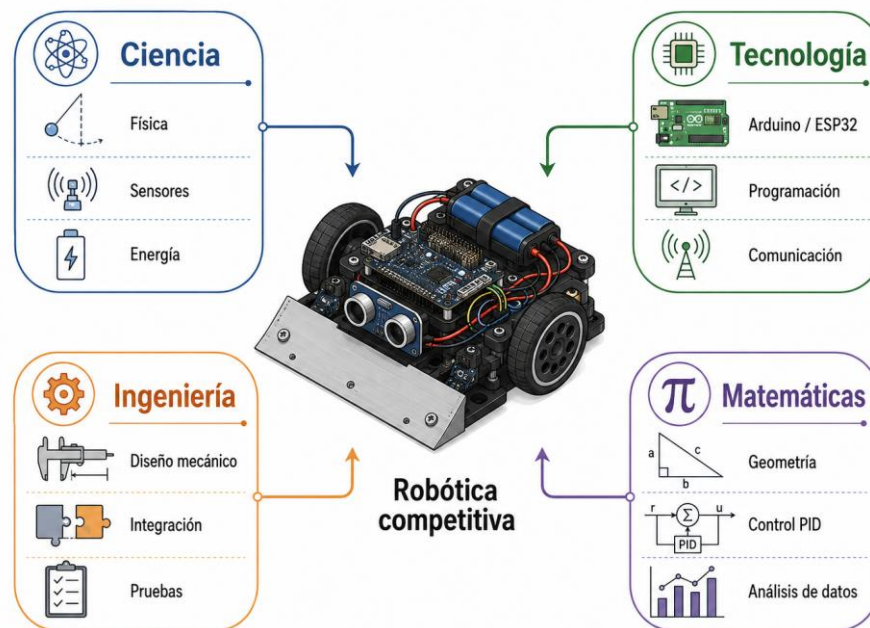


Figura 1.2: Relación entre la metodología STEM y la robótica competitiva.

1.5. Categorías principales de competencia

Las competencias de robótica se presentan en una amplia variedad de categorías, cada una con objetivos, reglamentos y desafíos particulares. Aunque las reglas específicas pueden variar entre eventos, la mayoría de las competiciones pueden agruparse en categorías ampliamente reconocidas a nivel nacional e internacional.

1.5.1. Minisumo

El Minisumo es una de las categorías más populares de la robótica competitiva. Su objetivo consiste en expulsar a un oponente fuera de una arena circular denominada dohyo. Generalmente, los robots deben cumplir restricciones de tamaño y peso, lo que obliga a optimizar cuidadosamente cada componente.

Las principales áreas de desarrollo incluyen estrategias de búsqueda, detección de adversarios, sistemas de tracción, distribución de peso, adherencia, control autónomo y protección ante el borde del dohyo.

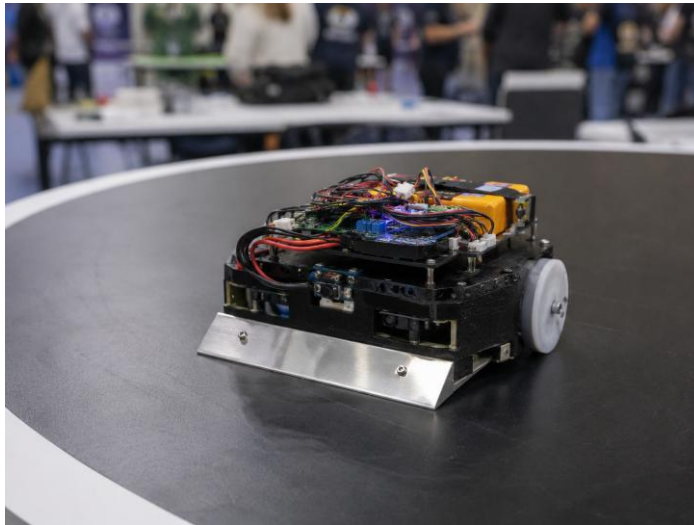


Figura 1.3: Robot Minisumo utilizado como plataforma de desarrollo y competencia.

1.5.2. Microsumo

El Microsumo representa una versión miniaturizada del Minisumo. Aunque las dimensiones son considerablemente menores, las exigencias técnicas suelen ser incluso mayores debido a las limitaciones de espacio, peso y energía. La miniaturización de los sistemas electrónicos constituye uno de los principales desafíos de esta categoría.

1.5.3. Seguidores de línea

Los robots seguidores de línea deben recorrer una pista en el menor tiempo posible utilizando sensores ópticos que detectan una trayectoria marcada sobre el suelo. Esta categoría constituye un excelente escenario para aplicar conceptos relacionados con control PID, procesamiento de señales, filtrado digital, ajuste de velocidad y optimización mecánica.

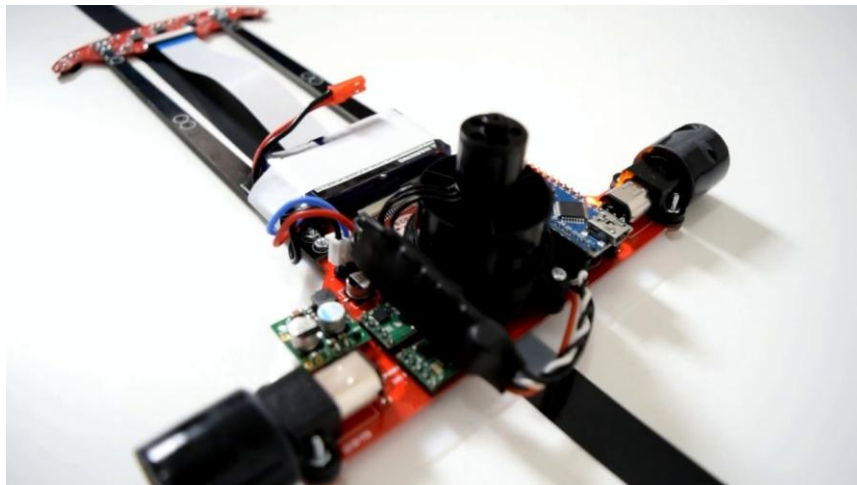


Figura 1.4: Robot seguidor de línea como plataforma para el estudio de control y sensores ópticos.

1.5.4. Robot Soccer

Los robots de fútbol deben localizar una pelota, desplazarse por una cancha delimitada y anotar goles de manera autónoma o semiautónoma. Esta categoría incorpora conceptos avanzados relacionados con navegación, visión artificial, estrategias colaborativas, comunicación entre robots y sistemas multi agente.

1.5.5. Robot Combate

Los robots de combate están diseñados para enfrentarse físicamente mediante mecanismos ofensivos y defensivos. Entre las tecnologías más utilizadas destacan armas rotativas, cuñas, sistemas de impacto, estructuras resistentes a choques y control remoto de alta confiabilidad. Debido a los riesgos asociados, esta categoría requiere estrictas medidas de seguridad mecánica, eléctrica y operativa.

1.5.6. Laberintos y exploración

En estas competencias los robots deben navegar por entornos desconocidos identificando rutas óptimas hasta alcanzar un objetivo. Las habilidades desarrolladas incluyen mapeo, localización, planeamiento de trayectorias y toma de decisiones autónoma [5].

1.6. Evolución tecnológica de los robots competitivos

La evolución de la robótica competitiva ha estado estrechamente vinculada al desarrollo de nuevas tecnologías electrónicas y computacionales.

1.6.1. Primera generación

Los primeros robots competitivos utilizaban circuitos discretos, lógica cableada, sensores básicos y controladores analógicos. Estos sistemas poseían capacidades limitadas y dependían en gran medida del diseño electrónico.

1.6.2. Segunda generación

La aparición de microcontroladores programables permitió incorporar algoritmos de control, sensores digitales, estrategias básicas de navegación y mayor flexibilidad de programación. Esta etapa marcó el inicio de la robótica embebida moderna.

1.6.3. Tercera generación

El desarrollo de plataformas como Arduino facilitó enormemente el acceso a la robótica educativa y competitiva. Los robots comenzaron a incorporar comunicación inalámbrica, control PID, sensores avanzados, telemetría y estructuras de software más organizadas.

1.6.4. Cuarta generación

Actualmente, los robots más avanzados integran microcontroladores con conectividad, visión artificial, inteligencia artificial, redes neuronales, procesamiento en la nube y sistemas colaborativos. El ESP32, por ejemplo, permite integrar conectividad WiFi y Bluetooth en sistemas de bajo costo, lo que amplía significativamente las posibilidades de desarrollo [6].



Figura 1.5: Evolución tecnológica de los robots competitivos.

1.7. Categorías principales de competencia

Las competencias internacionales desempeñan un papel fundamental en el avance de la robótica, ya que constituyen espacios donde se presentan nuevas tecnologías, metodologías y estrategias de diseño.

Entre los eventos más reconocidos se encuentran RoboCup, FIRST Robotics Competition, World Robot Olympiad y diferentes competencias especializadas de robótica móvil, sumo, combate y seguimiento de línea. RoboCup se define como una iniciativa científica internacional orientada a avanzar el estado del arte de los robots inteligentes [3]. FIRST Robotics Competition, por su parte, se presenta como un programa de robótica para estudiantes de secundaria donde los participantes desarrollan habilidades de ingeniería, trabajo en equipo y resolución de problemas [4].

Estos eventos constituyen importantes referentes para el desarrollo de nuevas tecnologías y estrategias de diseño robótico. Además, permiten que estudiantes y docentes contrasten sus propuestas con las de equipos de otros países, fortaleciendo la formación técnica y la cultura de innovación.

1.8. La robótica competitiva en Ecuador

Durante los últimos años, la robótica competitiva en Ecuador ha experimentado un crecimiento significativo gracias al trabajo conjunto de instituciones educativas, universidades, organizaciones tecnológicas y comunidades de entusiastas que han impulsado espacios de formación y competencia para estudiantes de diferentes niveles académicos.

Las primeras competencias nacionales estuvieron orientadas principalmente a categorías educativas y demostrativas. Sin embargo, con el paso del tiempo surgieron eventos cada vez más especializados que incorporaron categorías de alto desempeño similares a las desarrolladas en competencias internacionales.

Actualmente, diversas instituciones organizan eventos que incluyen categorías como Minisumo, Microsumo, Robot Soccer, seguidores de línea, batallas de robots, robótica educativa, desafíos de programación y automatización industrial. Estos eventos han permitido que estudiantes ecuatorianos desarrollen habilidades técnicas avanzadas y participen posteriormente en competencias internacionales.

Las universidades han desempeñado un papel fundamental en este proceso. A través de clubes de robótica, grupos de investigación y proyectos de vinculación, han logrado consolidar espacios permanentes para el desarrollo tecnológico.



Figura 1.6: Participación universitaria ecuatoriana en competencias de robótica.

1.9. Caso de estudio 1.1: Clubes universitarios de robótica

Los clubes universitarios de robótica constituyen espacios donde los estudiantes pueden complementar los conocimientos adquiridos en el aula mediante el desarrollo de proyectos reales. Entre las principales actividades desarrolladas por estos grupos destacan el diseño y construcción de robots, la participación en competencias nacionales e internacionales, el desarrollo de proyectos de investigación aplicada, la organización de talleres y la vinculación con instituciones educativas. La experiencia demuestra que los estudiantes que participan activamente en clubes de robótica desarrollan con mayor rapidez habilidades relacionadas con trabajo en equipo, liderazgo, resolución de problemas, gestión de proyectos e innovación tecnológica.

1.10. Arduino y ESP32 como plataformas de desarrollo

El crecimiento de la robótica competitiva ha estado estrechamente relacionado con la aparición de plataformas de hardware abierto que permiten desarrollar sistemas embebidos de forma rápida y económica. Entre las plataformas más utilizadas actualmente destacan Arduino y ESP32.

1.10.1. Arduino

Arduino es una plataforma de desarrollo de hardware y software abierto diseñada para facilitar la creación de sistemas electrónicos interactivos. Su popularidad se debe principalmente a su facilidad de programación, bajo costo, amplia documentación y gran disponibilidad de módulos y sensores. La placa Arduino UNO Rev3, por ejemplo, está basada en el microcontrolador ATmega328P, dispone de entradas y salidas digitales, entradas analógicas, comunicación USB y una frecuencia de operación de 16 MHz [1]. Aunque en muchos robots competitivos se utiliza Arduino Nano por su tamaño reducido, los conceptos de programación son equivalentes en las placas basadas en el mismo microcontrolador [7].

Tabla 1.1: Características generales de placas Arduino utilizadas en robótica educativa y competitiva.

Placa	Microcontrolador	Aplicaciones típicas
Arduino UNO	ATmega328P	Prototipado, prácticas educativas
Arduino Nano	ATmega328P	Minisumo, seguidores, instrumentación compacta
Arduino MEGA	ATmega2560	Proyectos con gran cantidad de entradas/salidas

1.10.2. ESP32

El ESP32 representa una evolución significativa respecto a las plataformas tradicionales de bajo costo. Además de sus capacidades de procesamiento, incorpora conectividad WiFi y Bluetooth, lo que permite desarrollar aplicaciones de telemetría, interfaces web, sistemas IoT y comunicación inalámbrica directa con otros dispositivos [2].

Tabla 1.2: Comparación general entre Arduino Nano y ESP32.

Característica	Arduino Nano	ESP32
Frecuencia típica	16MHz	Hasta 240 MHz
Conectividad WiFi	No	Sí
Bluetooth	No	Sí
Complejidad de uso	Baja	Media
Aplicación principal	Control embebido básico	IoT, telemetría, control avanzado

La elección de una u otra plataforma dependerá de los requerimientos específicos del proyecto. En este libro se utilizarán ambas tecnologías con el propósito de que el lector adquiera experiencia en diferentes escenarios de desarrollo.

1.11. Organización general del libro

El contenido de esta obra se encuentra organizado en seis partes principales:

- **Parte I. Fundamentos:** presenta los conceptos teóricos necesarios para comprender el funcionamiento de los sistemas embebidos y la robótica competitiva.
- **Parte II. Robótica competitiva:** incluye proyectos relacionados con Minisumo, telemetría, sistemas de seguridad y control de plataformas móviles
- **Parte III. Wearables tecnológicos:** desarrolla proyectos interactivos que combinan sensores, actuadores y comunicaciones inalámbricas.
- **Parte IV. Instrumentación inteligente:** introduce aplicaciones orientadas a adquisición de datos, monitoreo y análisis experimental.
- **Parte V. Ingeniería inversa y sistemas interactivos:** explora técnicas de comunicación, reconocimiento de señales y diseño de interfaces.
- **Parte VI. Control avanzado:** presenta aplicaciones relacionadas con perfiles de movimiento, control PID y telemetría web.

Actividades de aprendizaje

1. Investigue tres competencias internacionales de robótica y compare sus categorías principales.
2. Elabore una línea de tiempo que muestre la evolución de los sistemas robóticos desde los primeros controladores electrónicos hasta las plataformas actuales basadas en ESP32.
3. Identifique una problemática de su entorno que podría resolverse mediante la implementación de un sistema robótico.
4. Compare las características técnicas de Arduino Nano y ESP32 e identifique situaciones donde cada plataforma resulte más adecuada.
5. Diseñe un diagrama conceptual que relacione los cuatro componentes de la metodología STEM con la robótica competitiva.

Preguntas de reflexión

1. ¿Por qué la robótica competitiva constituye una herramienta efectiva para la enseñanza de disciplinas STEM?
2. ¿Cuáles son las principales diferencias entre un proyecto académico tradicional y un robot diseñado para competición?
3. ¿Qué ventajas ofrece el uso de plataformas de hardware abierto en el desarrollo de sistemas robóticos?
4. ¿Cómo contribuyen las competencias internacionales al avance tecnológico de la robótica?
5. ¿Qué competencias profesionales pueden desarrollarse mediante la participación en clubes de robótica?

Referencias

- [1] Arduino, “UNO R3”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/hardware/uno-rev3>. [Accedido: 24-jun-2026].
- [2] Espressif Systems, “ESP32 Series Datasheet”, Espressif Systems. [En línea]. Disponible en: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. [Accedido: 24-jun-2026].
- [3] RoboCup Federation, “RoboCup Federation official website”. [En línea]. Disponible en: <https://www.robotcup.org/>. [Accedido: 24-jun-2026].
- [4] FIRST, “FIRST Robotics Competition”, FIRST Inspires. [En línea]. Disponible en: <https://www.firstinspires.org/programs/frc/>. [Accedido: 24-jun-2026].
- [5] B. Siciliano and O. Khatib, Springer Handbook of Robotics, 2nd ed. Cham, Switzerland: Springer, 2016.
- [6] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, Introduction to Autonomous Mobile Robots, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.
- [7] M. Banzi and M. Shiloh, Getting Started with Arduino, 4th ed. Sebastopol, CA, USA: Maker Media, 2022.
- [8] A. Eguchi, “Educational robotics for promoting 21st century skills”, Journal of Automation, Mobile Robotics and Intelligent Systems, vol. 8, no. 1, pp. 5-11, 2014.

Capítulo 2. Electrónica para Robots Competitivos

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Identificar los principales subsistemas electrónicos presentes en robots competitivos y sistemas inteligentes.
- Seleccionar fuentes de alimentación, reguladores, sensores, actuadores y drivers de potencia según los requerimientos de una aplicación robótica.
- Comprender los riesgos asociados a baterías LiPo, caídas de voltaje, ruido eléctrico e interferencias electromagnéticas.
- Diferenciar las aplicaciones típicas de Arduino Nano y ESP32 en proyectos de robótica, instrumentación y telemetría.
- Aplicar criterios básicos de protección eléctrica para mejorar la confiabilidad y seguridad de un robot de competencia.

2.1. Introducción

La electrónica constituye uno de los pilares fundamentales de la robótica moderna. Independientemente de la categoría o complejidad del sistema desarrollado, todo robot requiere una infraestructura electrónica capaz de suministrar energía, procesar información, interactuar con sensores y controlar actuadores. En robótica competitiva, esta infraestructura debe ser además resistente, compacta, eficiente y confiable.

A diferencia de un prototipo académico diseñado únicamente para funcionar en condiciones controladas de laboratorio, un robot competitivo debe operar bajo escenarios exigentes. Durante una competencia puede estar expuesto a vibraciones, golpes, aceleraciones bruscas, cambios de iluminación, interferencias electromagnéticas, caídas de voltaje y variaciones de temperatura. Por ello, un diseño electrónico deficiente puede comprometer el desempeño del robot incluso cuando el software y la mecánica han sido correctamente desarrollados.

Un error común entre diseñadores principiantes consiste en concentrarse casi exclusivamente en la programación, asumiendo que el comportamiento del robot depende únicamente del código. Sin embargo, en la práctica muchas fallas observadas durante competencias tienen su origen en problemas eléctricos: baterías descargadas, reguladores subdimensionados, conexiones flojas, drivers sobrecalentados, ruido en sensores o mala distribución de tierra.

La electrónica aplicada a la robótica competitiva no debe entenderse solamente como la conexión de módulos comerciales. Implica tomar decisiones de diseño relacionadas con alimentación, protección, cableado, compatibilidad de señales, consumo energético, disipación térmica y seguridad del sistema.

Este capítulo presenta los conceptos fundamentales relacionados con la electrónica empleada en robots competitivos y sistemas inteligentes. Su propósito es proporcionar una base técnica que permita comprender los proyectos desarrollados posteriormente en el libro, especialmente aquellos relacionados con Minisumo, telemetría, control de motores, wearables, instrumentación y sistemas IoT.



Figura 2.1: Componentes electrónicos utilizados en el desarrollo de robots competitivos.

2.2. Arquitectura electrónica de un robot

Desde una perspectiva funcional, la mayoría de robots competitivos pueden dividirse en varios subsistemas electrónicos. Aunque la arquitectura final depende de la categoría, tamaño y propósito del robot, la estructura general suele incluir alimentación, control, sensores, actuadores, potencia y comunicación [7].

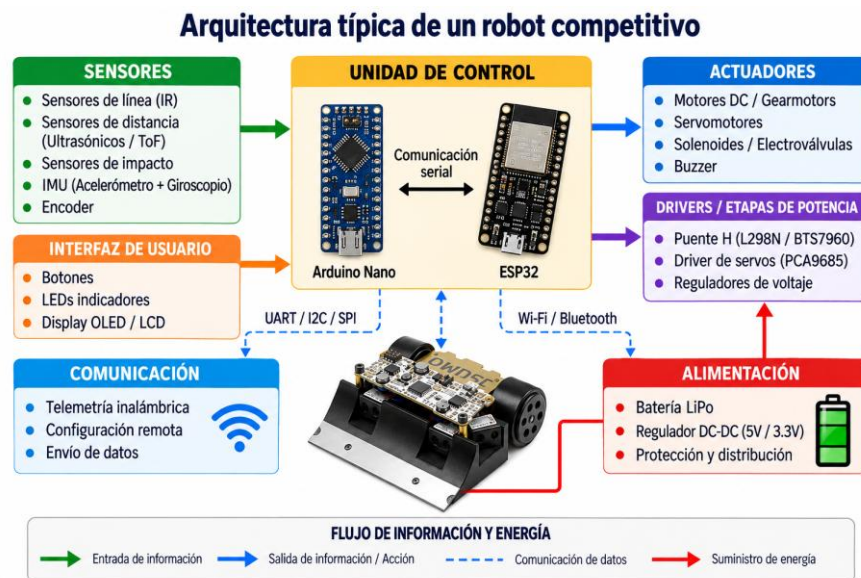


Figura 2.2: Arquitectura general de un robot competitivo.

Los subsistemas más comunes son:

- **Sistema de alimentación:** proporciona energía a todos los componentes del robot.
- **Sistema de control:** procesa señales, ejecuta algoritmos y toma decisiones.
- **Sistema de sensores:** obtiene información del entorno o del estado interno del robot.
- **Sistema de potencia:** controla motores y actuadores que demandan corrientes elevadas.
- **Sistema de comunicación:** permite la interacción con otros dispositivos, aplicaciones móviles, computadoras o redes inalámbricas.
- **Sistema de protección:** reduce riesgos eléctricos, térmicos y operativos.

Un robot competitivo debe diseñarse como un sistema integrado. No basta con que cada módulo funcione por separado; todos los subsistemas deben operar correctamente al mismo tiempo, compartiendo alimentación, señales y referencia de tierra de manera segura.

2.3. Sistema de alimentación

La alimentación eléctrica es uno de los elementos más críticos de cualquier robot. Una fuente de energía inadecuada puede provocar reinicios inesperados, pérdida de comunicación, disminución del rendimiento, falsas lecturas de sensores o daños permanentes en componentes electrónicos.

En robótica competitiva se busca una fuente de energía con:

- Voltaje adecuado para motores, drivers y electrónica de control.
- Corriente suficiente para soportar aceleraciones y picos de consumo.
- Bajo peso y tamaño reducido.
- Alta densidad energética.
- Capacidad de descarga compatible con la aplicación.
- Conectores seguros y baja resistencia interna.

2.3.1. Baterías LiPo

Las baterías de polímero de litio, conocidas comúnmente como LiPo, son ampliamente utilizadas en robótica competitiva debido a su alta densidad energética y capacidad de descarga. Cada celda LiPo posee un voltaje nominal aproximado de 3.7 V y un voltaje máximo de carga de 4.2 V.

Tabla 2.1: Valores típicos de voltaje por celda en baterías LiPo.

Estado de la celda	Voltaje aproximado
Descarga profunda no recomendada	Menor a 3.0 V
Zona baja de operación	3.2 V – 3.5 V
Voltaje nominal	3.7 V
Carga completa	4.2 V

Una batería de dos celdas en serie se identifica como 2S. Su voltaje nominal se calcula como:

$$V_{nominal} = 2 \times 3.7 = 7.4 \text{ V}$$

y su voltaje máximo completamente cargada es:

$$V_{max} = 2 \times 4.2 = 8.4 \text{ V}$$

2.3.2. Baterías LiPo HV

Las baterías LiPo HV (High Voltage) permiten un voltaje máximo por celda superior, típicamente 4.35 V. Esto proporciona mayor energía disponible, pero también puede llevar algunos sistemas electrónicos a operar cerca de sus límites.

Tabla 2.2: Comparación general entre LiPo convencional y LiPo HV.

Tipo de batería	Voltaje nominal por celda	Voltaje máximo por celda
LiPo convencional	3.7 V	4.20 V
LiPo HV	3.8 V aprox.	4.35 V

2.3.3. Caso de estudio 2.1: batería de 8 celdas en un robot de sumo

Considérese un robot de sumo que utiliza una batería de 8 celdas en serie. Si se emplea una batería LiPo convencional, el voltaje máximo será:

$$V_{max} = 8 \times 4.20 = 33.60 \text{ V}$$

Si se utiliza una batería LiPo HV, el voltaje máximo será:

$$V_{max} = 8 \times 4.35 = 34.80 \text{ V}$$

Aunque la diferencia puede parecer pequeña, en sistemas que utilizan controladores electrónicos diseñados para operar cerca de 35 V, este incremento puede resultar crítico. Esta situación muestra la importancia de verificar siempre el voltaje máximo real de una batería completamente cargada y compararlo con las especificaciones del driver, ESC, regulador o controlador utilizado.

Nunca se debe asumir que el voltaje nominal de una batería representa su valor máximo. Para seleccionar correctamente drivers, reguladores y protecciones, debe utilizarse el voltaje máximo de carga.



Figura 2.3: Baterías utilizadas en robots competitivos.

2.4. Regulación y distribución de energía

Los motores de un robot suelen requerir tensiones y corrientes diferentes a las utilizadas por sensores y microcontroladores. Por esta razón, es necesario implementar etapas de regulación y distribución de energía.

2.4.1. Reguladores lineales

Los reguladores lineales, como el 7805, reducen el voltaje de entrada disipando la energía excedente en forma de calor. Son simples y económicos, pero su eficiencia disminuye cuando la diferencia entre el voltaje de entrada y salida es elevada.

La potencia disipada puede estimarse mediante:

$$P_{disipada} = (V_{in} - V_{out})I$$

Por ejemplo, si un regulador lineal convierte 12 V a 5 V con una corriente de 500 mA:

$$P_{disipada} = (12 - 5)(0.5) = 3.5 \text{ W}$$

Esta potencia se transforma en calor, por lo que el regulador podría sobrecalentarse si no cuenta con disipación adecuada.

2.4.2. Convertidores DC – DC

Los convertidores DC-DC tipo *buck* son más eficientes que los reguladores lineales y se recomiendan para robots alimentados por baterías. Un módulo común utilizado en proyectos educativos es el LM2596, aunque para robots competitivos deben seleccionarse módulos de buena calidad y corriente suficiente.

Tabla 2.3: Comparación entre regulador lineal y convertidor DC-DC.

Característica	Regulador lineal	Convertidor DC – DC
Eficiencia	Baja a media	Alta
Complejidad	Baja	Media
Calentamiento	Alto si hay gran caída de voltaje	Menor
Uso recomendado	Corrientes pequeñas	Robots móviles y sistemas con batería

2.4.3. Buenas prácticas de distribución

Para mejorar la confiabilidad del sistema se recomienda:

- Separar físicamente cables de potencia y cables de señal.
- Utilizar conectores adecuados para la corriente esperada.
- Evitar protoboards en robots sometidos a vibraciones o impactos.
- Usar cables cortos y de calibre apropiado.
- Conectar todas las referencias de tierra necesarias de forma controlada.
- Añadir capacitores de desacoplo cerca de sensores y microcontroladores.

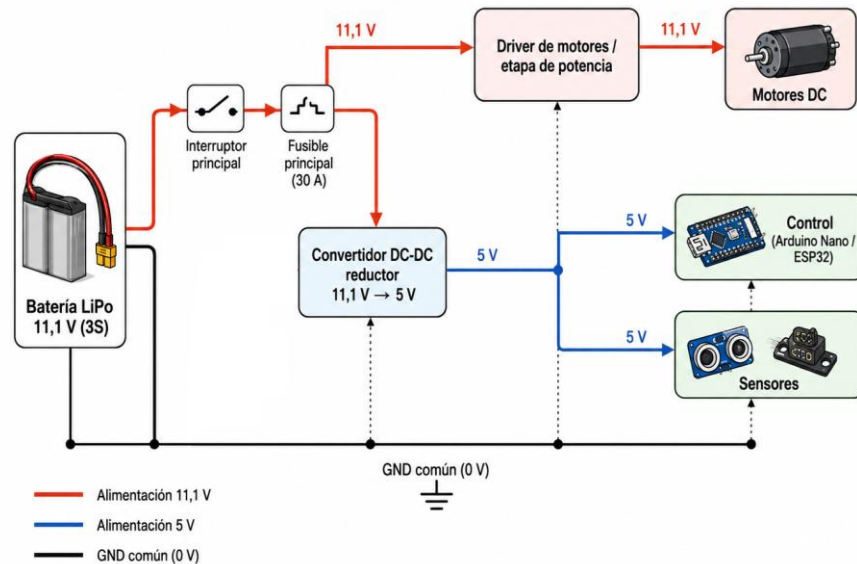


Figura 2.4: Distribución recomendada de energía en un robot móvil.

2.5. Microcontroladores

El microcontrolador actúa como el cerebro del robot. Su función es leer sensores, ejecutar algoritmos de control, generar señales para actuadores y comunicarse con otros dispositivos. En este libro se utilizarán principalmente Arduino Nano y ESP32, por su disponibilidad, bajo costo y utilidad educativa.

2.5.1. Arduino Nano

Arduino Nano es una placa compacta basada en el microcontrolador ATmega328P. Según la documentación oficial de Arduino, la placa Nano posee un formato reducido de 45 mm por 18 mm y está orientada a proyectos donde el espacio es importante [4]. Sus características la hacen adecuada para robots Minisumo, seguidores de línea, sistemas de instrumentación compactos y proyectos educativos [1].

Tabla 2.4: Características generales de Arduino Nano.

Parámetro	Valor general
Microcontrolador	ATmega328P
Frecuencia de reloj	16 MHz
Entradas analógicas	8
Memoria Flash	32 KB
EEPROM	1 KB
Aplicaciones típicas	Minisumo, seguidores, sensores, control básico

2.5.2. ESP32

El ESP32 es una familia de microcontroladores con conectividad inalámbrica integrada. Espressif describe al ESP32 como un chip combinado WiFi y Bluetooth de 2.4 GHz diseñado para aplicaciones de bajo consumo y sistemas conectados [2]. Sus módulos, como la serie ESP32-WROOM, son adecuados para aplicaciones que requieren telemetría, control remoto, interfaces web, IoT o comunicación con aplicaciones móviles [3].

Tabla 2.5: Características generales del ESP32.

Parámetro	Valor general
CPU	Doble núcleo en variantes clásicas
Frecuencia	Hasta 240 MHz en variantes comunes
Conectividad	WiFi y Bluetooth
Aplicaciones típicas	IoT, telemetría, dashboards, control avanzado
Complejidad	Media

2.5.3. Selección de plataforma

La selección entre Arduino Nano y ESP32 depende de los requerimientos del proyecto. Arduino Nano resulta conveniente cuando se necesita simplicidad, bajo consumo y control directo de sensores o motores. ESP32 es preferible cuando se requiere conectividad inalámbrica, mayor capacidad de procesamiento o integración con aplicaciones web.

Tabla 2.6: Criterios generales para seleccionar Arduino Nano o ESP32.

Criterio	Arduino Nano	ESP32
Facilidad inicial	Alta	Media
Conectividad integrada	No	Sí
Procesamiento	Bajo a medio	Medio a alto
Tamaño	Muy compacto	Compacto
Uso recomendado	Control embebido directo	IoT, telemetría, interfaces web

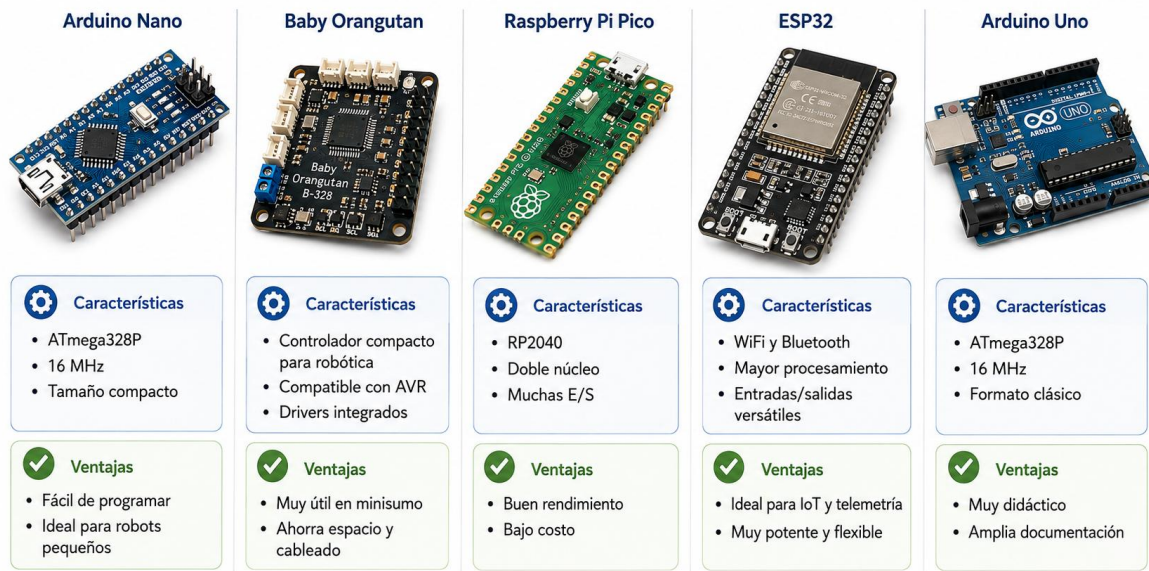


Figura 2.5: Plataformas de desarrollo utilizadas en los proyectos del libro.

2.6. Sensores utilizados en robótica competitiva

Los sensores permiten al robot percibir su entorno y conocer su propio estado. Una mala selección o calibración de sensores puede provocar comportamientos erráticos, pérdida de trayectoria, fallas de detección o decisiones incorrectas.

2.6.1. Sensores infrarrojos reflectivos

Los sensores infrarrojos reflectivos se utilizan para detectar diferencias de color o reflectancia en una superficie. Son comunes en seguidores de línea y robots de sumo para detectar el borde blanco del dohyo.

Ejemplos de sensores y barras utilizadas en robótica son TCRT5000, QTR y placas de sensores tipo Phoenix. Su funcionamiento se basa en emitir luz infrarroja y medir la cantidad de luz reflejada por la superficie.

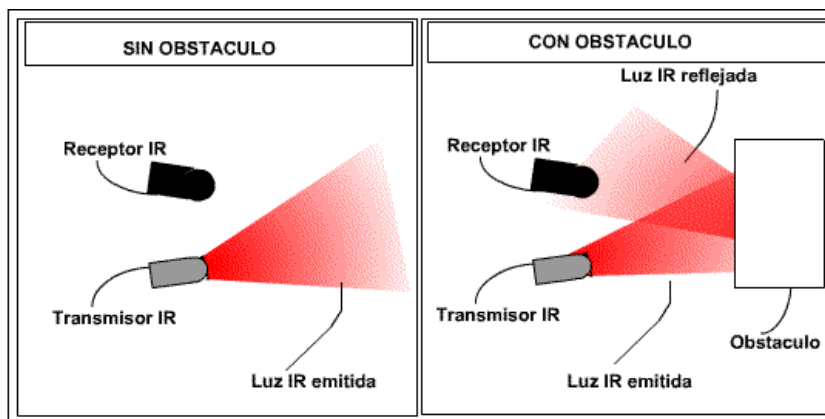


Figura 2.6: Principio de funcionamiento de un sensor infrarrojo reflectivo.

2.6.2. Sensores de distancia

Los sensores de distancia permiten detectar oponentes, obstáculos o paredes. En Minisumo se utilizan frecuentemente sensores infrarrojos tipo Sharp por su velocidad de respuesta y tamaño compacto. En proyectos educativos también se emplean sensores ultrasónicos como el HC-SR04, aunque su respuesta puede ser menos adecuada para escenarios de alta velocidad.

Para un sensor ultrasónico, la distancia puede estimarse mediante:

$$d = \frac{vt}{2}$$

donde d es la distancia, v la velocidad del sonido y t el tiempo de ida y retorno de la señal.

2.6.3. Preguntas de reflexión

Las unidades de medición inercial, como el MPU6050, permiten medir aceleración y velocidad angular. En robótica competitiva pueden utilizarse para registrar impactos, detectar inclinación, estimar vibraciones o analizar movimientos bruscos.

2.6.4. Referencias del capítulo

Los sensores eléctricos permiten supervisar variables internas del robot. Entre ellos destacan:

- Divisores resistivos para medición de voltaje.
- Sensores de corriente como ACS712.
- Sensores de temperatura para protección térmica.

El ACS712 es un sensor de corriente lineal aislado galvánicamente, utilizado en aplicaciones de medición de corriente AC o DC, control de motores y protección contra sobrecorriente [4, 5, 6].



Figura 2.7: Sensores utilizados en robots competitivos y sistemas inteligentes.

2.7. Actuadores

Los actuadores transforman energía eléctrica en movimiento, fuerza, sonido, luz o vibración. Su selección depende del tipo de tarea que debe realizar el robot.

2.7.1. Motores DC

Los motores DC con caja reductora son ampliamente utilizados en robots móviles debido a su simplicidad, disponibilidad y relación torque-peso. Se emplean en Minisumo, robots soccer, seguidores de línea y plataformas móviles [8].

2.7.2. Motores brushless

Los motores brushless ofrecen alta eficiencia, alta velocidad y buena relación potencia - peso. Se utilizan en drones, sistemas de succión, ventiladores de alto rendimiento y armas rotativas de robots de combate. Para su control requieren un ESC (Electronic Speed Controller).

2.7.3. Servomotores

Los servomotores permiten controlar posición angular. Son comunes en brazos robóticos, mecanismos articulados, wearables y sistemas de bloqueo.

2.7.4. Motores paso a paso

Los motores paso a paso se utilizan cuando se requiere posicionamiento preciso. Son comunes en impresoras 3D, máquinas CNC, sistemas de dosificación y mecanismos de posicionamiento.

2.7.5. Actuadores hápticos

Los motores vibradores y actuadores hápticos se utilizan para proporcionar retroalimentación táctil. En este libro aparecerán en proyectos de wearables, como el casco con radar háptico.



Figura 2.8: Actuadores utilizados en robótica competitiva, wearables e instrumentación.

2.8. Electrónica de potencia

Los pines de un microcontrolador no pueden alimentar directamente motores o cargas de alta corriente. Por ello se requieren etapas de potencia que actúen como intermediarias entre el sistema de control y los actuadores.

2.8.1. Puentes H

Un puente H permite controlar el sentido de giro de un motor DC. Drivers como TB6612FNG son utilizados ampliamente en robots pequeños y medianos. Toshiba identifica al TB6612FNG como un circuito integrado controlador de motores DC con dos canales [4]. Este tipo de driver es útil para robots Minisumo ligeros, seguidores de línea y plataformas educativas.

Driver	Tipo de carga	Aplicación típica
TB6612FNG	Motores DC pequeños	Minisumo, seguidores, robots educativos
L298N	Motores DC medianos	Prototipos educativos de baja eficiencia
BTS7960	Motores DC de mayor corriente	Robots móviles de mayor potencia
ESC	Motores brushless	Drones, armas rotativas, ventiladores

2.8.2. ESC para motores brushless

Un ESC convierte señales de control, generalmente PWM, en conmutación trifásica para motores brushless. En robots de combate y sistemas de alta potencia, el ESC debe seleccionarse considerando corriente máxima, voltaje de operación, capacidad térmica, tipo de firmware, respuesta dinámica y calidad de construcción.

El controlador de potencia siempre debe seleccionarse con margen de seguridad. Utilizar un driver o ESC al límite de sus especificaciones aumenta la probabilidad de sobrecalentamiento, reinicio o falla permanente durante una competencia.

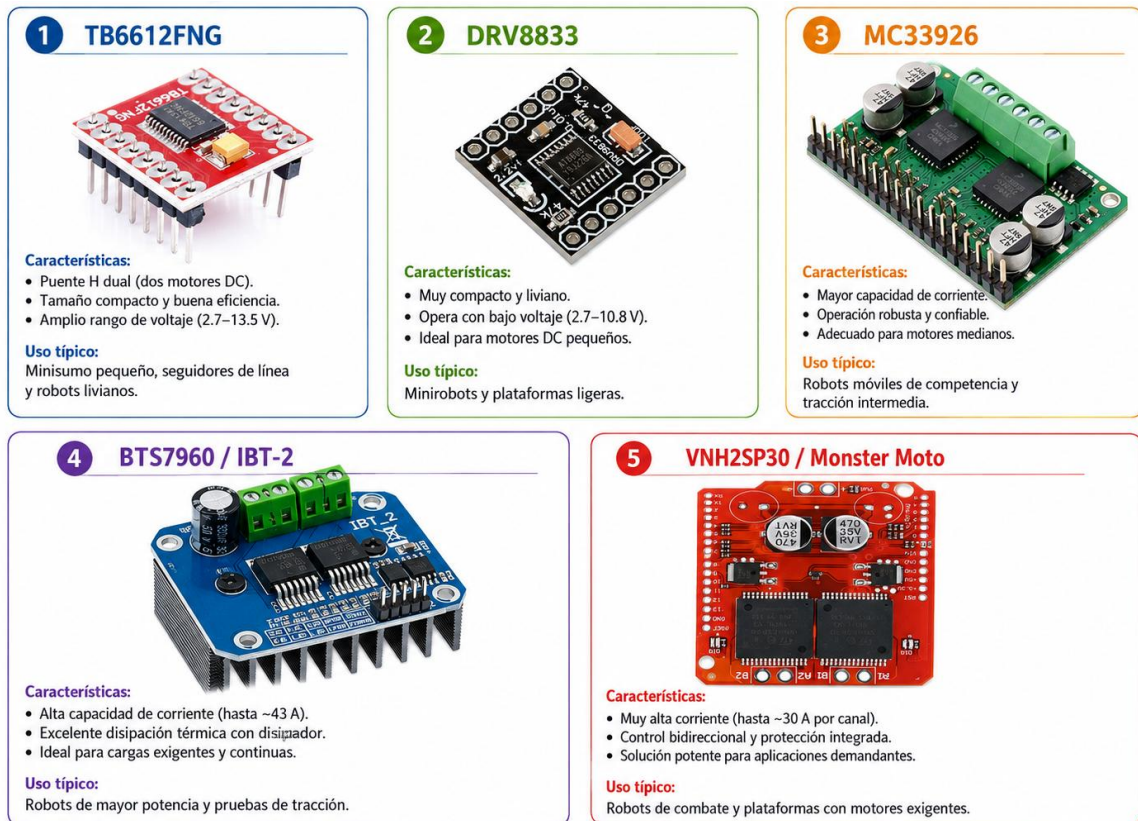


Figura 2.9: Etapas de potencia para motores DC y brushless.

2.9. Ruido eléctrico e interferencias

El ruido eléctrico es uno de los problemas más frecuentes en robots competitivos. Puede originarse en motores DC, ESCs, cables de potencia, convertidores DC-DC, transmisores inalámbricos o contactos deficientes.

Los efectos del ruido pueden incluir:

- Lecturas falsas en sensores.

- Reinicios del microcontrolador.
- Pérdida de comunicación.
- Activación accidental de entradas digitales.
- Fallas intermitentes difíciles de reproducir.

2.9.1. Técnicas de mitigación

Para reducir problemas de ruido se recomienda:

- Colocar capacitores de desacoplo cerca de módulos sensibles.
- Separar físicamente cables de potencia y cables de señal.
- Usar cables trenzados cuando sea conveniente.
- Evitar lazos de tierra innecesarios.
- Fijar correctamente conectores y módulos.
- Utilizar reguladores de calidad.
- Proteger líneas críticas mediante resistencias, filtros o aislamiento cuando sea necesario.

2.10. Protección eléctrica y seguridad

La seguridad eléctrica es esencial en robots competitivos. Un sistema sin protección puede causar daños en componentes, pérdida de control o situaciones peligrosas para los operadores.

2.10.1. Fusibles e interruptores

Los fusibles protegen contra cortocircuitos y sobrecorrientes. El interruptor general permite desconectar rápidamente el sistema durante pruebas o emergencias.

2.10.2. Protección contra inversión de polaridad

La inversión accidental de polaridad puede destruir microcontroladores, sensores y drivers. Se pueden utilizar diodos, MOSFETs de protección o conectores polarizados para reducir este riesgo.

2.10.3. Protección contra sobretensiones

Los diodos TVS pueden utilizarse para limitar picos de voltaje generados por cargas inductivas o eventos transitorios.

2.10.4. Supervisión de batería

En sistemas alimentados por LiPo es importante monitorear el voltaje para evitar descargas profundas. Esto puede implementarse mediante divisores resistivos conectados a entradas analógicas del microcontrolador.

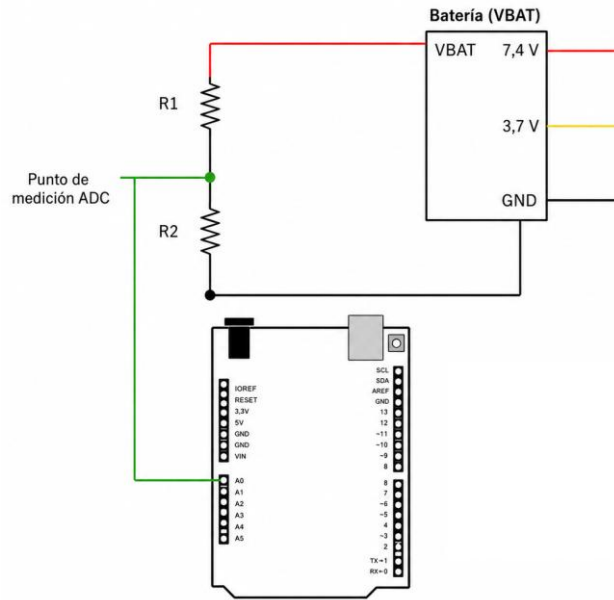


Figura 2.10: Medición de voltaje de batería mediante divisor resistivo.

El voltaje en el ADC se calcula mediante:

$$V_{ADC} = V_{BAT} \left(\frac{R_2}{R_1 + R_2} \right)$$

Por tanto, el voltaje real de la batería puede recuperarse como:

$$V_{BAT} = V_{ADC} \left(\frac{R_1 + R_2}{R_2} \right)$$

2.11. Caso de estudio 2.2: electrónica de un robot Minisumo

Un robot Minisumo competitivo puede organizarse mediante la siguiente arquitectura:

- Batería LiPo 2S.
- Regulador DC - DC para electrónica de control.
- Arduino Nano como microcontrolador principal.
- Sensores de borde infrarrojos.
- Sensores frontales de distancia.
- Driver TB6612FNG para motores DC.
- Motores con caja reductora.
- Interruptor general y conectores seguros.

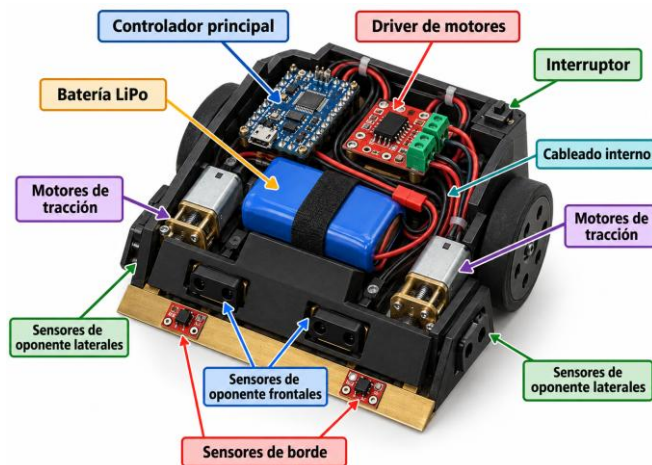


Figura 2.11: Distribución electrónica interna de un robot Minisumo.

2.12. Caso de estudio 2.3: electrónica de un robot de combate

Un robot de combate incorpora requerimientos más exigentes debido a los impactos y corrientes elevadas. Su arquitectura puede incluir:

- Batería LiPo de alta capacidad de descarga.
- Receptor RC.
- ESCs para motores de tracción.
- ESC para arma brushless. Sistema Fail-Safe.
- Interruptor de seguridad.
- Protección mecánica para los componentes electrónicos.

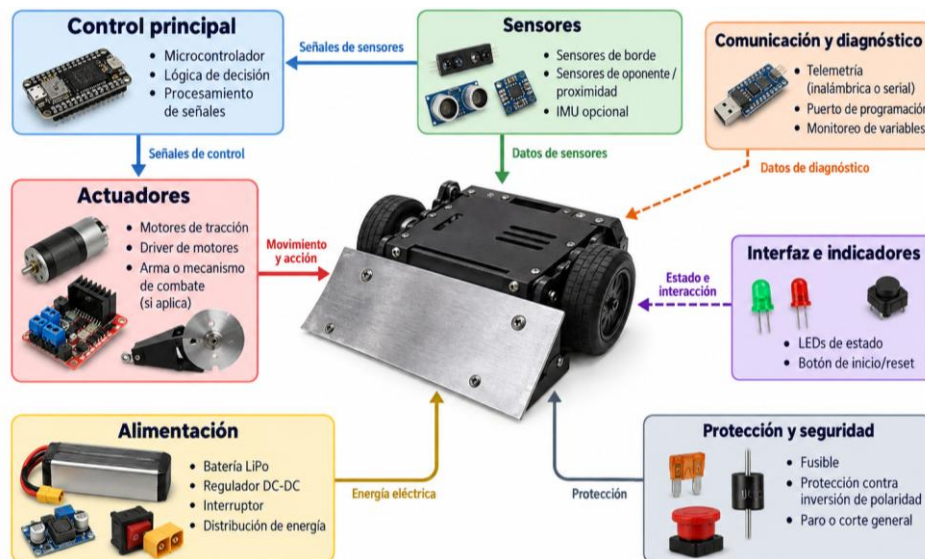


Figura 2.12: Subsistemas electrónicos típicos de un robot de combate.

2.13. Buenas prácticas de montaje

Para aumentar la confiabilidad de un robot competitivo se recomienda:

1. Diseñar primero la arquitectura electrónica antes de conectar módulos.
2. Verificar polaridad y voltajes con multímetro antes de energizar.
3. Probar sensores y actuadores por separado.
4. Usar conectores adecuados para cada nivel de corriente.
5. Fijar físicamente baterías, drivers y microcontroladores.
6. Evitar conexiones sueltas o cables excesivamente largos [9].
7. Documentar el cableado mediante diagramas y etiquetas.
8. Realizar pruebas de vibración y movimiento antes de competir.

Un robot no debe probarse por primera vez en el evento. La electrónica debe validarse previamente mediante pruebas de consumo, calentamiento, vibración, autonomía y respuesta ante fallas.

Actividades de aprendizaje

1. Calcule el voltaje nominal y máximo de una batería LiPo 3S, 4S y 6S.
2. Compare las ventajas y desventajas de Arduino Nano y ESP32 para un robot Minisumo.
3. Diseñe la arquitectura electrónica básica de un robot seguidor de línea de alta velocidad.
4. Investigue tres drivers de motores DC y compare su voltaje, corriente y aplicación recomendada.
5. Diseñe un divisor resistivo para medir una batería 3S con una entrada analógica de 5 V.
6. Proponga tres estrategias para reducir ruido eléctrico en un robot con motores DC.

Preguntas de reflexión

1. ¿Por qué una batería debe seleccionarse considerando su voltaje máximo y no solo su voltaje nominal?
2. ¿Qué ventajas ofrece un convertidor DC-DC frente a un regulador lineal en robots móviles?
3. ¿Por qué los motores no deben conectarse directamente a los pines de un microcontrolador?
4. ¿Qué fallas pueden aparecer cuando no se separan adecuadamente potencia y control?
5. ¿Qué criterios utilizaría para seleccionar un driver para un robot de competencia?

Referencias

[1] Arduino, “Nano”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/hardware/nano>. [Accedido: 24-jun-2026].

- [2] Espressif Systems, “ESP32 Series Datasheet”, Espressif Systems. [En línea]. Disponible en: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. [Accedido: 24-jun-2026].
- [3] Espressif Systems, “ESP32 Wi-Fi & Bluetooth Modules”, Espressif Systems. [En línea]. Disponible en: <https://www.espressif.com/en/products/modules/esp32>. [Accedido: 24-jun-2026].
- [4] Toshiba Electronic Devices & Storage Corporation, “TB6612FNG Brush Motor Driver IC”, Toshiba. [En línea]. Disponible en: <https://toshiba.semicon-storage.com/ap-en/semiconductor/product/motor-driver-ics/brushed-dc-motor-driver-ics/detail.TB6612FNG.html>. [Accedido: 24-jun-2026].
- [5] Allegro MicroSystems, “ACS712 Galvanically Isolated Linear Current Sensor IC”, Allegro MicroSystems. [En línea]. Disponible en: <https://www.allegromicro.com/en/products/sense/current-sensor-ics/integrated-current-sensors/acs712>. [Accedido: 24-jun-2026].
- [6] Allegro MicroSystems, “ACS712 Datasheet”, Allegro MicroSystems. [En línea]. Disponible en: <https://www.allegromicro.com/-/media/files/datasheets/acs712-datasheet.ashx>. [Accedido: 24-jun-2026].
- [7] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed. Cham, Switzerland: Springer, 2016.
- [8] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.
- [9] D. Cook, *Robot Building for Beginners*, 4th ed. Berkeley, CA, USA: Apress, 2020.

Capítulo 3. Arquitectura de Software para Sistemas Embebidos

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender la estructura general de un programa embebido desarrollado para Arduino y ESP32.
- Diferenciar entre programación bloqueante y programación no bloqueante.
- Diseñar máquinas de estados para controlar el comportamiento de robots y sistemas inteligentes.
- Aplicar interrupciones, temporizadores y señales PWM en aplicaciones de robótica competitiva.
- Utilizar comunicación serial, Bluetooth, WiFi y almacenamiento no volátil en sistemas embebidos.
- Reconocer la importancia de la modularidad, la documentación y las buenas prácticas de programación.

3.1. Introducción

La evolución de la robótica moderna ha estado estrechamente ligada al desarrollo de sistemas embebidos cada vez más potentes, compactos y accesibles. Aunque la electrónica proporciona la infraestructura física necesaria para el funcionamiento de un robot, es el software quien determina su comportamiento, su capacidad de adaptación y su nivel de inteligencia.

En un robot competitivo, el software es responsable de interpretar información proveniente de sensores, ejecutar algoritmos de decisión y generar señales de control que permitan alcanzar los objetivos definidos por la estrategia de operación. Un robot Minisumo, por ejemplo, debe detectar al adversario, evitar salir del dohyo, tomar decisiones de ataque y recuperarse de situaciones desfavorables en fracciones de segundo. Un robot seguidor de línea debe leer continuamente una barra de sensores, calcular el error respecto a la trayectoria y ajustar la velocidad de sus motores con suficiente rapidez para no perder la pista.

A diferencia de las aplicaciones desarrolladas para computadoras personales, los sistemas embebidos trabajan bajo restricciones estrictas de procesamiento, memoria, energía y tiempo de respuesta. Estas limitaciones obligan a utilizar metodologías de programación orientadas a la eficiencia, confiabilidad y claridad estructural. El programa de un robot no debe ser solamente funcional; debe ser comprensible, mantenible y capaz de responder adecuadamente ante eventos inesperados.

En robótica competitiva, donde las decisiones deben ejecutarse en milisegundos, la arquitectura del software adquiere una importancia crítica. Un algoritmo eficiente puede representar la

diferencia entre ganar o perder una competencia. De igual manera, una mala estructura de programación puede provocar retrasos, bloqueos, pérdida de datos, respuestas tardías o comportamientos impredecibles.

Este capítulo presenta los principios fundamentales de programación utilizados en los proyectos desarrollados a lo largo del libro. Se abordan conceptos como estructura de programas Arduino, programación no bloqueante, máquinas de estados, interrupciones, PWM, comunicación serial, Bluetooth, WiFi, EEPROM, filtros digitales, control PID y buenas prácticas de desarrollo.



Figura 3.1: Desarrollo de software embebido aplicado a robótica competitiva.

3.2. Sistemas embebidos en robótica

Un sistema embebido es un sistema computacional diseñado para ejecutar una o varias funciones específicas dentro de un dispositivo electrónico. A diferencia de una computadora de propósito general, un sistema embebido opera normalmente con recursos limitados y se encuentra integrado directamente dentro del producto o prototipo que controla [6, 7].

En robótica, los sistemas embebidos se utilizan para:

- Leer sensores de distancia, borde, luz, corriente, temperatura o movimiento.
- Procesar información y tomar decisiones.
- Generar señales para motores, servomotores, ESCs y actuadores.
- Comunicarse con aplicaciones móviles, computadoras o módulos inalámbricos.
- Registrar datos de funcionamiento para análisis posterior.
- Supervisar condiciones de seguridad.

Arduino Nano y ESP32 son ejemplos de plataformas ampliamente utilizadas en sistemas embebidos educativos y competitivos. Arduino destaca por su simplicidad y bajo costo, mientras que ESP32 ofrece mayor capacidad de procesamiento y conectividad inalámbrica integrada [2, 4].

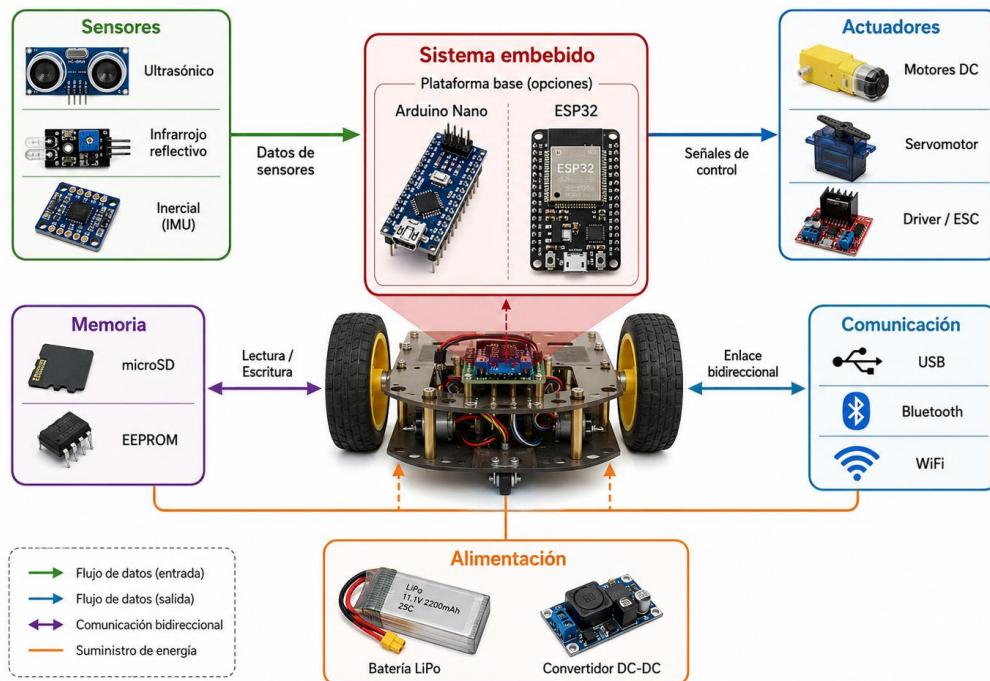


Figura 3.2: Ubicación del sistema embebido dentro de una plataforma robótica.

3.3. Estructura básica de un programa Arduino

Un programa desarrollado en el entorno Arduino se denomina comúnmente sketch. Su estructura mínima está compuesta por dos funciones principales: `setup()` y `loop()` [1, 8].

```
void setup()
{
  // Código que se ejecuta una sola vez
}
void loop()
{
  // Código que se ejecuta continuamente
}
```

3.3.1. Función `setup`

La función `setup()` se ejecuta una sola vez al encender o reiniciar el microcontrolador. Se utiliza para configurar pines, inicializar sensores, establecer comunicaciones, definir modos de operación y preparar variables iniciales.

```
void setup()
{
  Serial.begin(115200);
  pinMode(13, OUTPUT);
}
```


3.3.2. Función loop

La función `loop()` se ejecuta de manera indefinida mientras el sistema permanezca energizado. En ella se implementa el ciclo principal de operación del sistema.

```
void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

Aunque este ejemplo es útil para comprender la estructura básica, no representa una buena práctica para robots competitivos debido al uso de la función `delay()`, como se analizará en la siguiente sección.

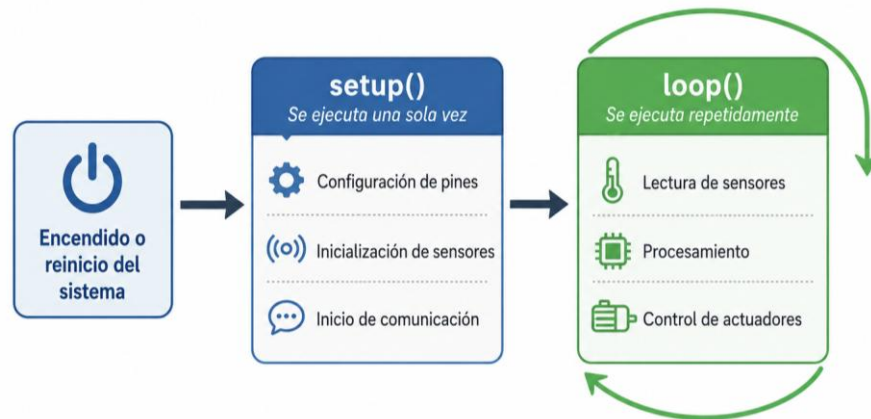


Figura 3.3: Ciclo básico de ejecución de un programa Arduino.

3.4. Variables y tipos de datos

La selección adecuada de tipos de datos permite optimizar el uso de memoria y mejorar la eficiencia del programa. Esto es especialmente importante en plataformas con recursos limitados como Arduino Nano.

Tabla 3.1: Tipos de datos comunes en Arduino.

Tipo	Tamaño aproximado	Uso típico
bool	1 byte	Estados lógicos
byte	1 byte	Valores de 0 a 255
int	2 bytes en AVR	Contadores y valores enteros
long	4 bytes	Tiempos y conteos grandes
float	4 bytes	Valores decimales

Ejemplo:

```
byte velocidadBase = 120;
float distanciaFrontal = 25.4;
bool oponenteDetectado = false;
unsigned long tiempoAnterior = 0;
```

En sistemas embebidos pequeños se debe evitar el uso innecesario de variables de gran tamaño. Un programa que parece funcionar correctamente en una computadora puede consumir demasiada memoria en un microcontrolador de recursos limitados.

3.5. Programación bloqueante

Uno de los errores más frecuentes en proyectos iniciales de Arduino es el uso excesivo de la función `delay()`. Esta función detiene la ejecución del programa durante el tiempo especificado.

```
digitalWrite(LED, HIGH);
delay(1000);
digitalWrite(LED, LOW);
delay(1000);
```

Durante el tiempo definido por `delay()`, el microcontrolador no ejecuta otras tareas dentro del ciclo principal. En aplicaciones simples esto puede ser aceptable; sin embargo, en robótica competitiva puede generar problemas graves.

Consecuencias del uso excesivo de `delay()`:

- Lectura tardía de sensores.
- Pérdida de eventos rápidos.
- Respuesta lenta ante el borde del dohyo.
- Comunicación interrumpida o poco fluida.
- Dificultad para ejecutar varias tareas simultáneamente.

3.6. Caso de estudio 3.1: retraso crítico en Minisumo

Considérese un robot Minisumo que utiliza un retraso de un segundo para ejecutar una maniobra de búsqueda. Durante ese intervalo, el robot podría no detectar al oponente, no percibir el borde del dohyo y continuar avanzando hasta salir de la pista. Esta situación evidencia por qué los robots competitivos deben evitar bloqueos innecesarios.

3.7. Programación no bloqueante con `millis`

La programación no bloqueante permite que el microcontrolador ejecute varias tareas dentro del ciclo principal sin detener completamente el programa. Para ello se utiliza normalmente la función `millis()`, que devuelve el tiempo transcurrido desde que el sistema fue energizado.


```

unsigned long tiempoAnterior = 0;
bool estadoLED = false;

void loop()
{
  if (millis() - tiempoAnterior >= 1000)
  {
    tiempoAnterior = millis();
    estadoLED = !estadoLED;
    digitalWrite(13, estadoLED);
  }
  leerSensores();
  controlarMotores();
  enviarTelemetria();
}

```

La principal ventaja de este enfoque es que el sistema continúa ejecutando otras funciones mientras espera que se cumpla una condición temporal.

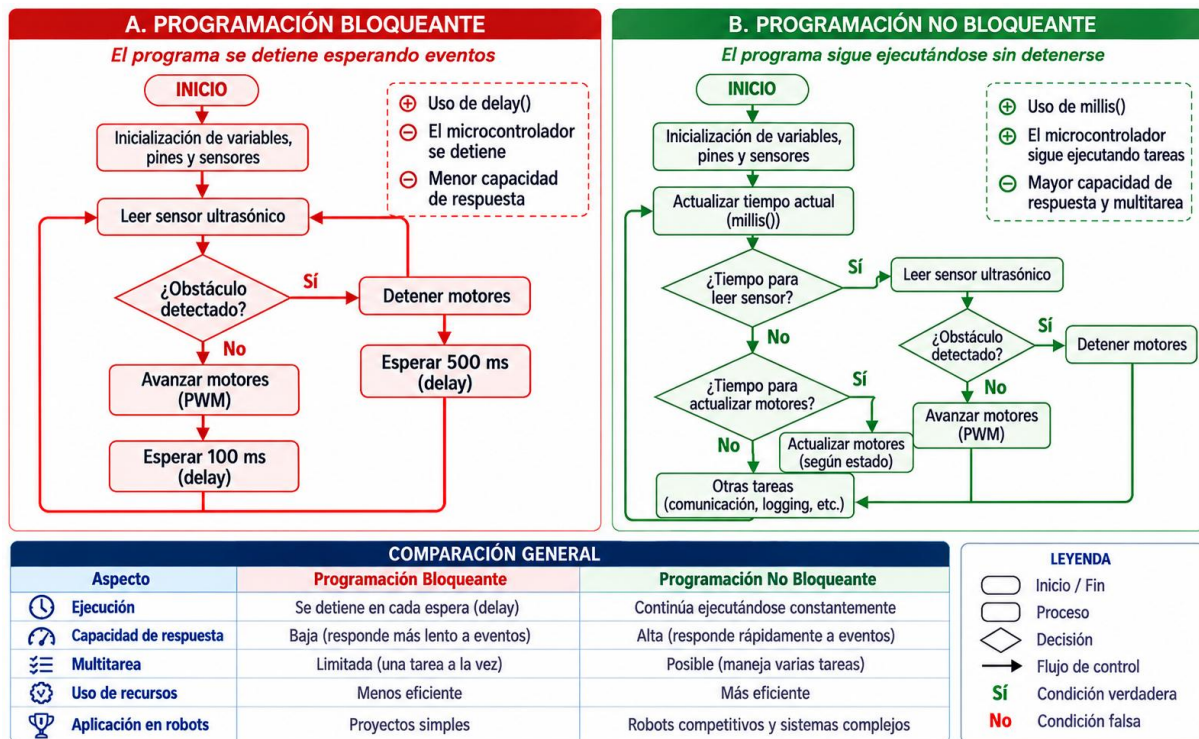


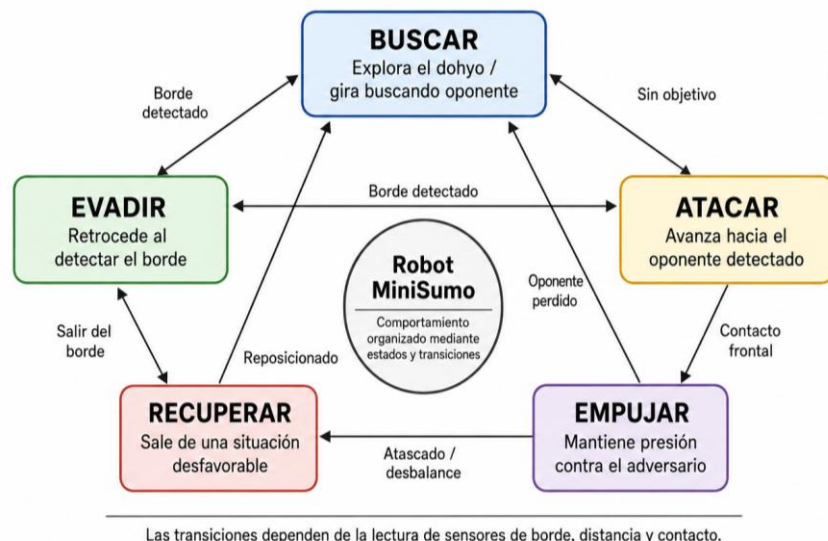
Figura 3.4: Comparación entre programación bloqueante y no bloqueante.

3.8. Máquinas de estados

Las máquinas de estados finitos constituyen una de las herramientas más importantes para organizar el comportamiento de robots y sistemas inteligentes. Permiten representar el programa como un conjunto de estados, acciones y transiciones.

Un robot Minisumo, por ejemplo, puede operar en los siguientes estados:

- **Buscar:** el robot gira o explora el dohyo.
- **Atacar:** el robot avanza hacia el oponente detectado.
- **Empujar:** el robot mantiene presión contra el adversario.
- **Evadir:** el robot retrocede al detectar el borde.
- **Recuperar:** el robot ejecuta una maniobra para salir de una situación desfavorable.



Las transiciones dependen de la lectura de sensores de borde, distancia y contacto.

Figura 3.5: Máquina de estados para un robot Minisumo.

La implementación en Arduino puede realizarse mediante un tipo enumerado:

```
enum EstadoRobot
{
    BUSCAR,
    ATACAR,
    EMPUJAR,
    EVADIR,
    RECUPERAR
};

EstadoRobot estadoActual = BUSCAR;
```

Posteriormente, el comportamiento se organiza mediante una estructura switch:

```
void loop()
{
  leerSensores();

  switch (estadoActual)
  {
    case BUSCAR:
      ejecutarBusqueda();
      break;

    case ATACAR:
      ejecutarAtaque();
      break;

    case EMPUJAR:
      ejecutarEmpuje();
      break;

    case EVADIR:
      ejecutarEvasion();
      break;
    case RECUPERAR:
      ejecutarRecuperacion();
      break;
  }
}
```

Las máquinas de estados permiten que un robot actúe de forma organizada. En lugar de escribir una gran cantidad de condiciones dispersas, el comportamiento se divide en estados claros y transiciones verificables.

3.9. Funciones y modularidad

La modularidad consiste en dividir un programa complejo en partes más pequeñas, comprensibles y reutilizables. En robótica competitiva esta práctica es fundamental porque facilita el mantenimiento del código y reduce errores durante la etapa de pruebas.

```
void leerSensores()
{
  // Lectura de sensores de borde y distancia
}

void calcularEstrategia()
{
  // Actualizacion de estado del robot
}
```

```

}

void controlarMotores()
{
    // Envio de velocidad a los motores
}

void enviarTelemetria()
{
    // Envio de datos por serial o Bluetooth
}

```

Un programa modular es más fácil de revisar, depurar y mejorar. Además, permite reutilizar funciones en distintos proyectos.

3.10. Interrupciones

Las interrupciones permiten que el microcontrolador responda de manera inmediata ante un evento externo o interno, sin esperar a que el ciclo principal llegue a una instrucción específica [2].

Aplicaciones comunes en robótica:

- Conteo de pulsos de encoders.
- Lectura de señales PWM de receptores RC.
- Medición de frecuencia.
- Detección de eventos rápidos.

Ejemplo básico:

```

volatile long pulsos = 0;

void setup()
{
    pinMode(2, INPUT);
    attachInterrupt(digitalPinToInterrupt(2), contarPulso, RISING);
}

void contarPulso()
{
    pulsos++;
}

```

Dentro de una rutina de interrupción se recomienda:

- Mantener el código lo más corto posible.
- No utilizar delay().
- No imprimir datos mediante Serial.
- Usar variables volatile cuando sean compartidas con el ciclo principal.

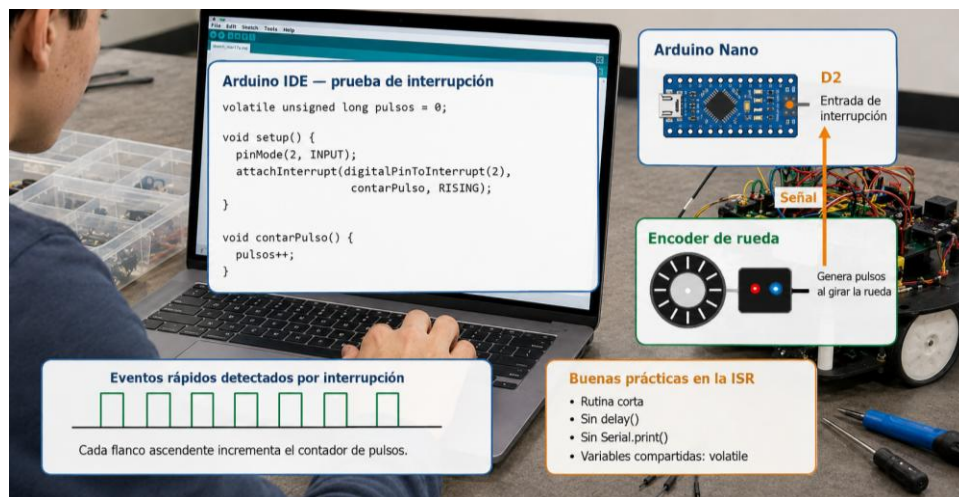


Figura 3.6: Uso de interrupciones para lectura de eventos rápidos en robótica.

3.11. PWM

PWM significa Pulse Width Modulation o modulación por ancho de pulso. Esta técnica permite controlar la potencia promedio entregada a una carga mediante una señal digital que alterna entre encendido y apagado.

En robótica, PWM se utiliza para:

- Controlar velocidad de motores DC mediante drivers.
- Regular intensidad de LEDs.
- Enviar señales a servomotores y ESCs.
- Controlar actuadores proporcionalmente.

En Arduino, una señal PWM básica puede generarse con:

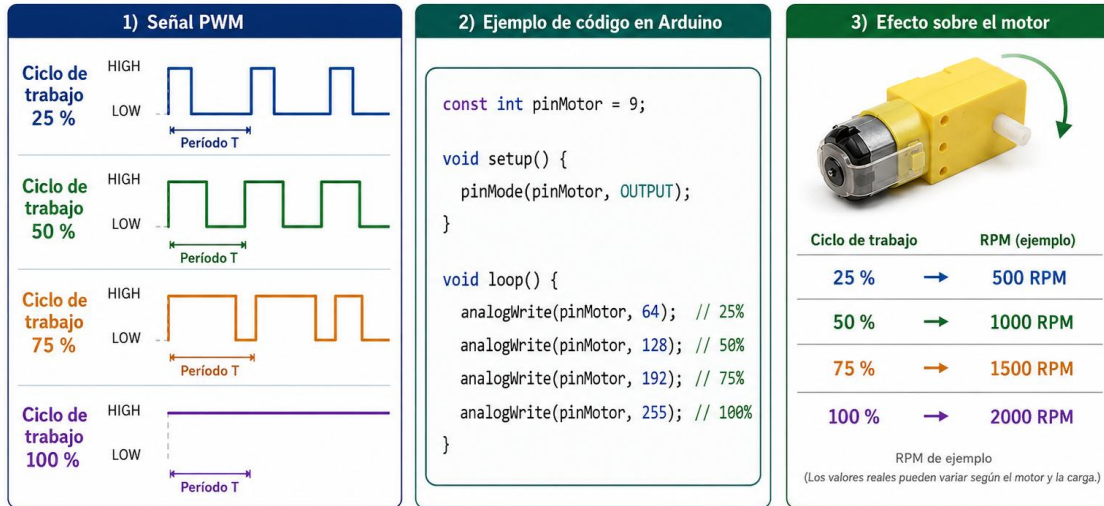
```
analogWrite(9, 128);
```

En placas Arduino basadas en AVR, el valor de analogWrite() normalmente se encuentra entre 0 y 255.

Tabla 3.2: Relación aproximada entre valor PWM y ciclo de trabajo.

Valor PWM	Ciclo de trabajo aproximado	Interpretación
0	0 %	Apagado
64	25 %	Baja potencia
128	50 %	Potencia media
192	75 %	Potencia alta
255	100 %	Máxima potencia

Misma amplitud y período; varía el tiempo que la señal permanece en nivel alto.



$$\text{Ciclo de trabajo (\%)} = (\text{TON} / T) \times 100$$



A mayor ciclo de trabajo, mayor potencia promedio entregada al motor.

Figura 3.7: Representación conceptual de una señal PWM.

3.12. Comunicación serial

La comunicación serial permite intercambiar datos entre el microcontrolador y otros dispositivos. Es una herramienta fundamental para depuración, telemetría y comunicación con módulos externos.

Inicialización:

```
Serial.begin(115200);
```

Envío de datos:

```
Serial.print("Distancia: ");
Serial.println(distancia);
```

Recepción de datos:

```
if (Serial.available())
{
  char dato = Serial.read();
}
```

En los proyectos de instrumentación, la comunicación serial permitirá enviar datos hacia MATLAB para su visualización y análisis.

3.13. Comunicación Bluetooth y aplicaciones móviles

Bluetooth permite establecer comunicación inalámbrica entre un microcontrolador y un teléfono inteligente, una computadora u otro sistema embebido. En proyectos basados en Arduino es común utilizar módulos como HC-05 o HC-06. En ESP32, la conectividad Bluetooth se encuentra integrada en el propio microcontrolador.

Aplicaciones típicas:

- Control remoto de robots.
- Telemetría hacia App Inventor.
- Configuración de parámetros.
- Activación de modos de operación.

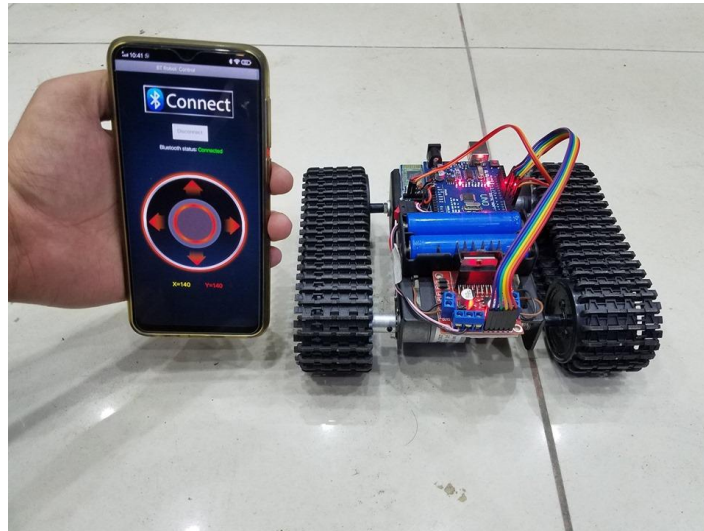


Figura 3.8: Interfaz móvil para control y telemetría mediante Bluetooth.

3.14. Comunicación WiFi con ESP32

El ESP32 permite desarrollar aplicaciones conectadas mediante WiFi. Esta característica será utilizada en proyectos del libro relacionados con telemetría, monitoreo remoto, dashboards web y ajuste inalámbrico de parámetros [3].

Un ejemplo conceptual de conexión WiFi en ESP32 es:

```
#include <WiFi.h>

const char* ssid = "NombreRed";
const char* password = "ClaveRed";

void setup()
{
```



```

Serial.begin(115200);

WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED)
{
    delay(500);
    Serial.print(".");
}

Serial.println("Conectado a WiFi");
}

```

Aunque el ejemplo utiliza un pequeño retraso dentro de la fase de conexión, en sistemas críticos se recomienda implementar mecanismos no bloqueantes o estados de conexión controlados.

3.15. Almacenamiento no volátil

En muchas aplicaciones es necesario conservar información incluso después de apagar el sistema. Para ello se utilizan memorias no volátiles como EEPROM, memoria flash o tarjetas SD [4].

Usos comunes:

- Guardar parámetros PID.
- Registrar eventos de falla.
- Almacenar récords de juego.
- Conservar configuraciones de usuario.
- Guardar datos experimentales.

Ejemplo básico con EEPROM:

```

#include <EEPROM.h>
void setup()
{
    EEPROM.write(0, 120);
    byte valor = EEPROM.read(0);
}

```

En proyectos avanzados, el almacenamiento debe gestionarse cuidadosamente para evitar escrituras excesivas y desgaste prematuro de memoria.

3.16. Filtros digitales

Los sensores reales rara vez entregan señales perfectas. Ruido eléctrico, vibraciones, variaciones ambientales y tolerancias de fabricación pueden provocar lecturas fluctuantes. Para mejorar la calidad de la señal se utilizan filtros digitales.

3.16.1. Promedio móvil

El promedio móvil calcula la media de un conjunto de muestras recientes.

$$\bar{x} = \frac{x_1 + x_2 + \dots + x_n}{n}$$

Es simple, fácil de implementar y útil para señales con ruido aleatorio moderado.

```
float promedio = (m1 + m2 + m3 + m4 + m5) / 5.0;
```

3.16.2. Promedio móvil

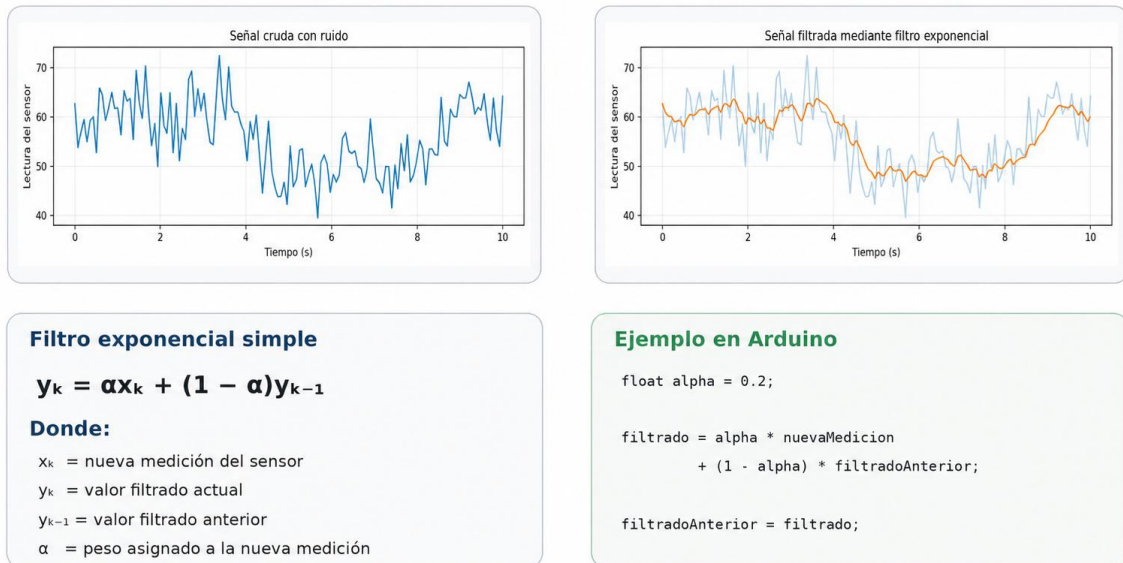
El filtro exponencial combina la nueva medición con el valor filtrado anterior.

$$y_k = \alpha x_k + (1 - \alpha)y_{k-1}$$

donde α determina qué tanto peso se otorga a la nueva medición.

```
float alpha = 0.2;
filtrado = alpha * nuevaMedicion + (1 - alpha) * filtradoAnterior;
filtradoAnterior = filtrado;
```

Los filtros digitales aparecerán posteriormente en proyectos relacionados con sensores de borde, simulación de fallas, telemetría de impactos e instrumentación.



El filtrado reduce las variaciones rápidas producidas por ruido y conserva la tendencia general de la señal.

Figura 3.9: Efecto conceptual de un filtro digital sobre una señal de sensor.

3.17. Introducción al control PID

El control PID es uno de los algoritmos más utilizados en robótica y automatización. Su objetivo es reducir el error entre una variable deseada y una variable medida [5].

La forma general del controlador PID es:

$$u(t) = K_p e(t) + K_i \int e(t) dt + K_d \frac{de(t)}{dt}$$

donde:

- K_p corrige el error actual.
- K_i corrige el error acumulado.
- K_d anticipa cambios y ayuda a reducir oscilaciones.

Aplicaciones en robótica:

- Seguidores de línea.
- Control de velocidad.
- Robots balanceadores.
- Control de posición.
- Sistemas térmicos.

3.18. Caso de estudio 3.2: seguidor de línea de alta velocidad

En un robot seguidor de línea de alta velocidad, una ganancia proporcional demasiado elevada puede provocar oscilaciones excesivas y pérdida de la trayectoria. Por el contrario, una ganancia derivativa correctamente ajustada ayuda a estabilizar el movimiento y mejorar el desempeño en curvas. Este tipo de ajuste requiere pruebas reales, análisis experimental y comprensión del comportamiento dinámico del robot.

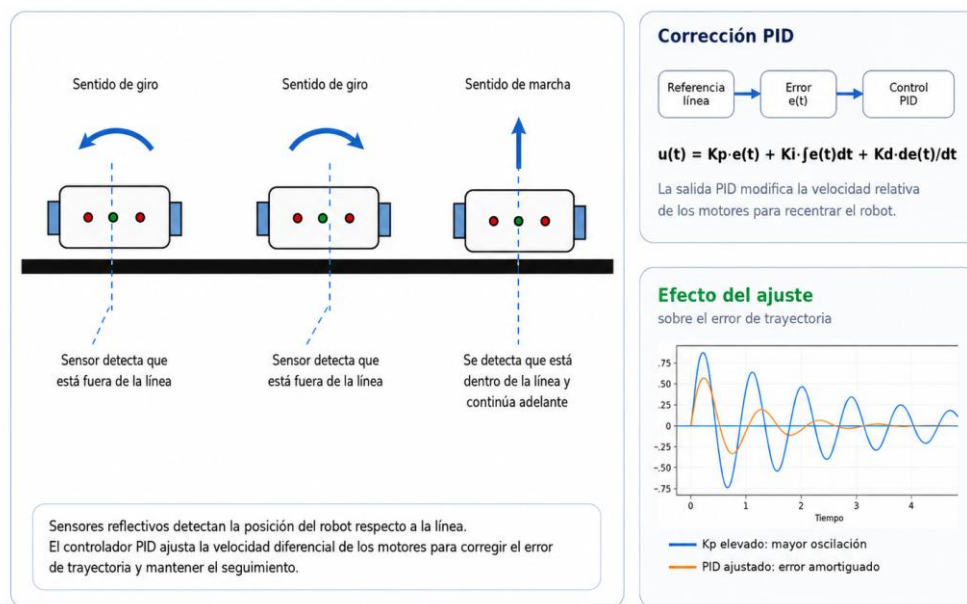


Figura 3.10: Aplicación del control PID en un robot seguidor de línea.

3.19. Buenas prácticas de programación embebida

Para desarrollar software confiable en robótica se recomienda aplicar las siguientes prácticas:

1. Utilizar nombres descriptivos para variables y funciones.
2. Evitar retrasos bloqueantes innecesarios.
3. Dividir el programa en funciones pequeñas.
4. Documentar entradas, salidas y parámetros importantes.
5. Usar máquinas de estados para comportamientos complejos.
6. Probar sensores y actuadores por separado antes de integrarlos.
7. Registrar datos de depuración durante las pruebas.
8. Implementar mecanismos de seguridad ante fallas.
9. Evitar código duplicado.
10. Mantener una estructura de carpetas ordenada.

3.20. Caso de estudio 3.3: arquitectura de software para Minisumo

Un robot Minisumo competitivo puede organizar su software en módulos funcionales:

- **Lectura de sensores:** sensores de borde y sensores de oponente.
- **Procesamiento:** filtrado, umbrales y detección de eventos.
- **Estrategia:** máquina de estados del robot.
- **Control de motores:** generación de señales PWM para el driver.
- **Seguridad:** detección de fallas o condiciones críticas.
- **Telemetría:** envío de información para depuración.

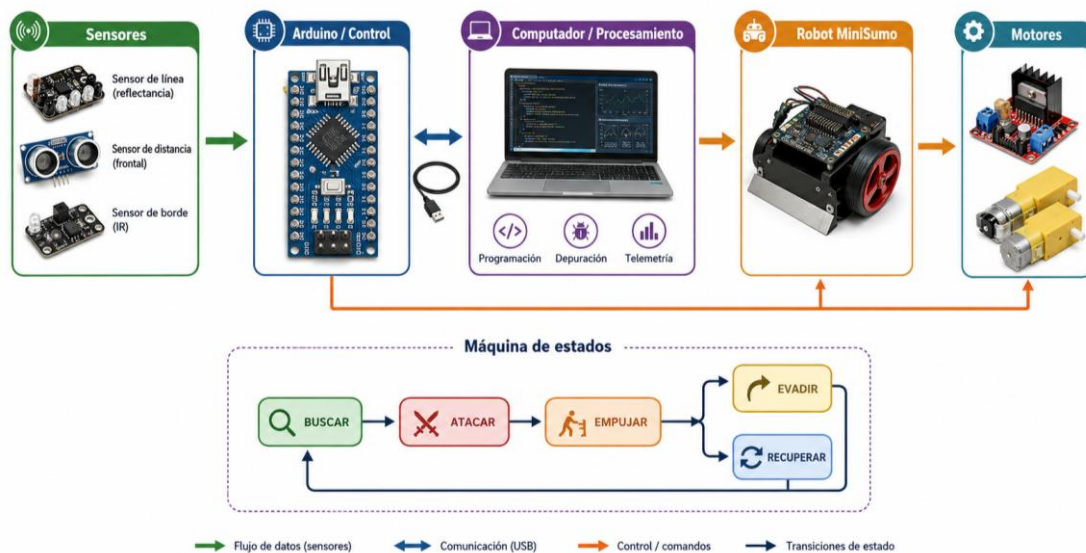


Figura 3.11: Arquitectura modular de software para un robot Minisumo.

Actividades de aprendizaje

1. Convertir un programa basado en `delay()` a una versión utilizando `millis()`.
2. Diseñar una máquina de estados para un semáforo inteligente.
3. Implementar una interrupción para contar pulsos de un encoder.
4. Desarrollar un control de velocidad mediante PWM para un motor DC con driver.
5. Guardar y recuperar un parámetro desde EEPROM.
6. Diseñar la arquitectura modular de software para un robot seguidor de línea.

Preguntas de reflexión

1. ¿Por qué las máquinas de estados son ampliamente utilizadas en robótica?
2. ¿Qué ventajas ofrece la programación no bloqueante frente al uso de `delay()`?
3. ¿Cuándo resulta conveniente utilizar interrupciones?
4. ¿Qué problemas pueden surgir por el uso excesivo de variables globales?
5. ¿Cuál es la importancia del control PID en aplicaciones robóticas?
6. ¿Por qué la modularidad facilita el desarrollo de proyectos complejos?

Referencias

- [1] Arduino, “Arduino Language Reference”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/language-reference/>. [Accedido: 24-jun2026].
- [2] Arduino, “attachInterrupt()”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/language-reference/en/functions/external-interrupts/attachInterrupt/>. [Accedido: 24-jun-2026].
- [3] Espressif Systems, “Arduino Core for ESP32”, Espressif Systems GitHub Documentation. [En línea]. Disponible en: <https://docs.espressif.com/projects/arduino-esp32/en/latest/>. [Accedido: 24-jun-2026].
- [4] Arduino, “EEPROM Library”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/learn/built-in-libraries/eeprom/>. [Accedido: 24- jun-2026].
- [5] K. J. Astrom and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*, 2nd ed. Princeton, NJ, USA: Princeton University Press, 2021.
- [6] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed. Cham, Switzerland: Springer, 2016.
- [7] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.
- [8] J. Boxall, *Arduino Workshop*, 2nd ed. San Francisco, CA, USA: No Starch Press, 2021.

Capítulo 4. Supervisor Inteligente de Seguridad para Robots Competitivos

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Diseñar un sistema embebido de supervisión para robots competitivos alimentados por baterías.
- Medir voltaje de batería mediante divisor resistivo y entrada analógica.
- Medir corriente mediante un sensor Hall tipo ACS712.
- Detectar pérdida o congelamiento de una señal de control RC.
- Implementar una lógica de seguridad tipo fail-safe para deshabilitar actuadores ante condiciones críticas.
- Enviar datos de telemetría por comunicación serial o Bluetooth para análisis posterior.
- Organizar el proyecto mediante una arquitectura modular clara, separando medición, seguridad, telemetría y actuación.

4.1. Introducción

La seguridad es un elemento esencial en cualquier sistema robótico que utilice baterías, motores, drivers de potencia, actuadores de alta velocidad o comunicación inalámbrica. En robótica competitiva, esta necesidad se vuelve aún más evidente debido a las condiciones extremas en las que operan los robots: aceleraciones bruscas, impactos, consumo elevado de corriente, vibraciones, interferencias electromagnéticas y cambios rápidos de estado [9].

Un robot puede perder el control por múltiples razones. La batería puede descargarse por debajo de un nivel seguro, un motor puede bloquearse y elevar el consumo de corriente, un driver puede sobrecalentarse, una señal de radiofrecuencia puede perderse o el microcontrolador principal puede quedar bloqueado por un error de software. En cualquiera de estos casos, el sistema debe ser capaz de entrar en un estado seguro.

El presente capítulo desarrolla un proyecto denominado *Supervisor Inteligente de Seguridad para Robots Competitivos*. Su propósito es implementar un módulo auxiliar basado en Arduino Nano capaz de monitorear variables críticas del robot y actuar preventivamente ante fallas. Este módulo puede ser incorporado en robots Minisumo, robots de combate, plataformas móviles RC, seguidores de línea de alta potencia o sistemas experimentales que requieran supervisión eléctrica.

El proyecto se organiza mediante una arquitectura modular, separando claramente las funciones de medición, seguridad, telemetría, almacenamiento y actuación. Esta forma de trabajo facilita la comprensión del sistema, permite realizar pruebas por etapas y reduce errores durante la

integración. Además, introduce una filosofía importante para el desarrollo de robots competitivos: un robot de alto desempeño no solo debe ser rápido o potente; también debe ser seguro, medible y confiable [3, 7].



Figura 4.1: Robot competitivo como plataforma de integración del supervisor inteligente.

4.2. Planteamiento del problema

En competencias de robótica es frecuente observar fallas asociadas a la alimentación y al control de potencia. Algunas de las situaciones más comunes son:

- Pérdida de fuerza por batería descargada.
- Reinicios del microcontrolador por caídas de voltaje.
- Sobrecalentamiento de drivers o ESCs.
- Corrientes elevadas por motores bloqueados.
- Lecturas erráticas de sensores debido a ruido eléctrico.
- Pérdida de señal RC o Bluetooth.
- Comportamiento impredecible del robot después de un impacto.

Muchas de estas fallas no son visibles para el operador hasta que el robot deja de funcionar. Por esta razón, resulta conveniente incorporar un sistema de supervisión que permita medir variables críticas y tomar decisiones automáticas de protección.

El objetivo del supervisor inteligente no es reemplazar al controlador principal del robot, sino complementarlo. Su función es monitorear variables críticas y ordenar un estado seguro cuando las condiciones de operación superan límites definidos.

4.3. Descripción general del proyecto

El sistema propuesto utiliza un Arduino Nano como microcontrolador de supervisión [1].

Este recibe información de tres fuentes principales:

- Voltaje de batería mediante divisor resistivo.
- Corriente consumida mediante sensor ACS712.
- Señal de control RC mediante entrada digital con interrupción.

A partir de estas mediciones, el sistema evalúa si el robot se encuentra en condición normal, advertencia o falla. Cuando se detecta una condición crítica, el supervisor activa una salida de seguridad que puede utilizarse para deshabilitar el driver de motores, cortar una señal de habilitación, activar una alarma o informar al controlador principal.

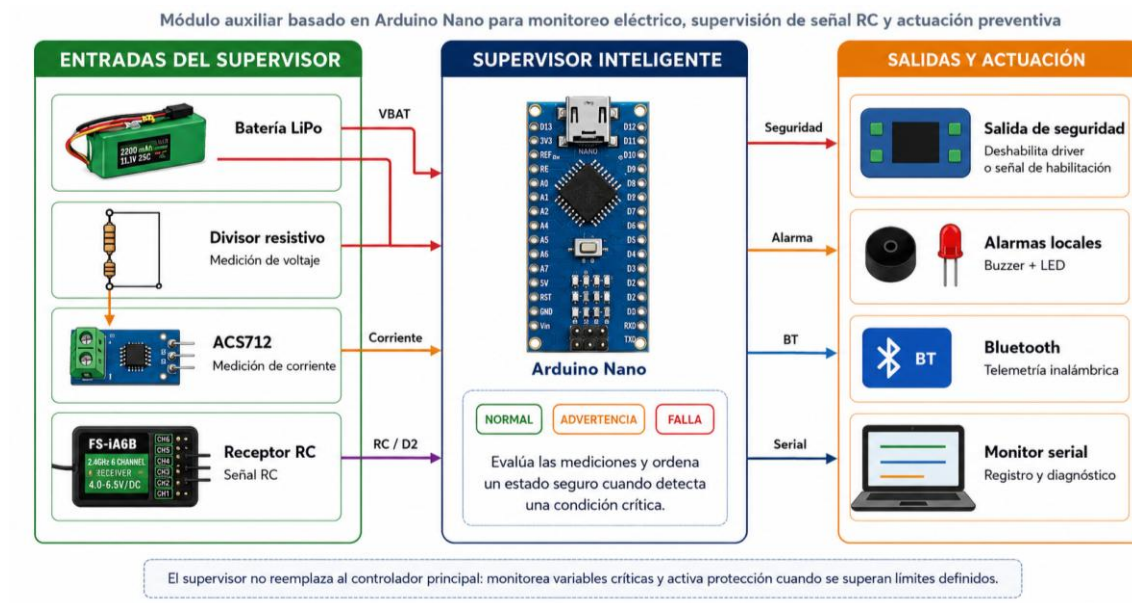


Figura 4.2: Arquitectura general del supervisor inteligente de seguridad.

4.4. Aplicación STEM

Este proyecto integra las cuatro dimensiones de la metodología STEM:

- **Ciencia:** análisis de corriente, voltaje, potencia, temperatura y efectos de descarga en baterías.
- **Tecnología:** uso de microcontroladores, sensores de corriente, comunicación serial y módulos Bluetooth.
- **Ingeniería:** diseño de un sistema de protección aplicable a robots reales de competencia.
- **Matemáticas:** cálculo de divisores resistivos, conversión ADC-voltaje, umbrales, promedios y estimación de potencia.

4.5. Objetivos del proyecto

4.5.1. Objetivo general

Diseñar e implementar un sistema embebido de supervisión para robots competitivos, capaz de monitorear voltaje, corriente y señal de control, generando acciones de seguridad ante condiciones críticas.

4.5.2. Objetivos específicos

1. Implementar medición de voltaje de batería mediante divisor resistivo.
2. Implementar medición de corriente mediante sensor ACS712.
3. Detectar pérdida de señal RC mediante interrupción y temporización.
4. Activar una salida de seguridad ante batería baja, sobrecorriente o pérdida de señal.
5. Enviar telemetría por comunicación serial o Bluetooth.
6. Registrar eventos básicos de falla en EEPROM.

4.6. Materiales y herramientas

Tabla 4.1: Materiales principales para el supervisor inteligente de seguridad.

Elemento	Cantidad	Función
Arduino Nano	1	Microcontrolador de supervisión
Sensor ACS712	1	Medición de corriente
Resistencias para divisor	2	Medición de voltaje de batería
Módulo Bluetooth HC-05	1	Telemetría opcional
Buzzer activo	1	Alarma sonora
LED indicador	1	Estado visual del sistema
Resistencia 220 Ω para LED	1	Limitación de corriente para LED
Cables y conectores	Varios	Interconexión
Batería LiPo	1	Fuente del robot
Driver o ESC	1 o más	Etapas de potencia a supervisar

4.7. Fundamento teórico

4.7.1. Medición de voltaje con divisor resistivo

Las entradas analógicas de Arduino Nano permiten medir voltajes dentro del rango permitido por el microcontrolador. Debido a que una batería de robot puede superar los 5 V, es necesario reducir el voltaje mediante un divisor resistivo.

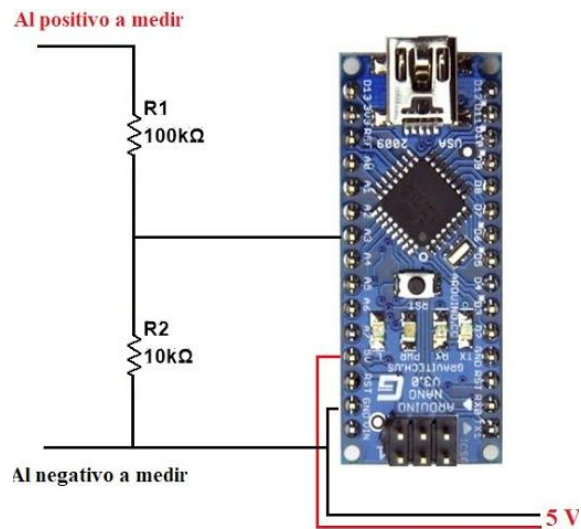


Figura 4.3: Divisor resistivo para medición de voltaje de batería.

El voltaje medido por el ADC se calcula como:

$$V_{ADC} = V_{BAT} \left(\frac{R_2}{R_1 + R_2} \right)$$

Por tanto, el voltaje de batería puede obtenerse mediante:

$$V_{BAT} = V_{ADC} \left(\frac{R_1 + R_2}{R_2} \right)$$

Para una batería 3S, cuyo voltaje máximo es aproximadamente 12.6 V, se puede utilizar una relación de resistencias que reduzca este valor por debajo de 5 V. Por ejemplo, con $R_1 = 30 \text{ k}\Omega$ y $R_2 = 10 \text{ k}\Omega$:

$$V_{ADC} = 12.6 \left(\frac{10}{30 + 10} \right) = 3.15V$$

Este valor se encuentra dentro del rango seguro para la entrada analógica de Arduino.

4.7.2. Medición de corriente con ACS712

El ACS712 es un sensor de corriente basado en efecto Hall. Permite medir corriente continua o alterna manteniendo aislamiento galvánico entre la línea de potencia y la salida de medición [4, 5]. Su salida es analógica y se encuentra centrada alrededor de la mitad del voltaje de alimentación.

En un sistema alimentado a 5 V, cuando la corriente es cero, la salida típica se encuentra cerca de 2.5 V. Cuando circula corriente, la salida aumenta o disminuye dependiendo del sentido de circulación.

La corriente puede estimarse mediante:

$$I = \frac{V_{OUT} - V_{ZERO}}{S}$$

donde:

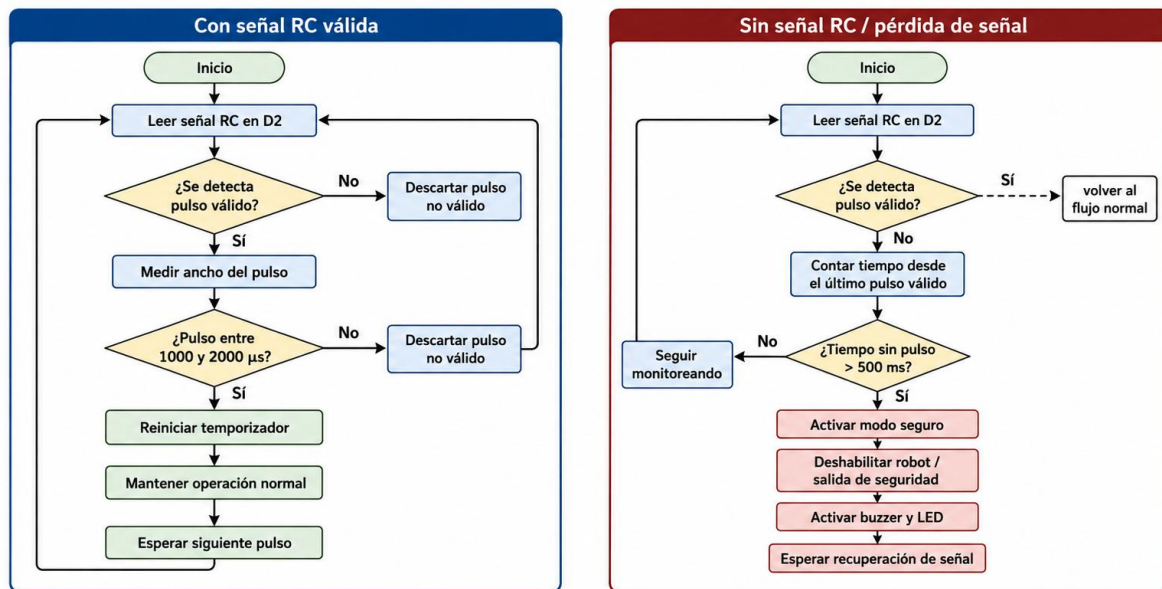
- I es la corriente medida.
- V_{OUT} es el voltaje de salida del sensor.
- V_{ZERO} es el voltaje de salida sin corriente.
- S es la sensibilidad del sensor.

La sensibilidad depende de la versión del módulo. Valores típicos son 185 mV/A para 5 A, 100 mV/A para 20 A y 66 mV/A para 30 A.

4.7.3. Detección de señal RC

En robots controlados por radio, el receptor suele entregar señales PWM a los controladores o microcontroladores. Una señal RC típica posee pulsos entre 1000 y 2000 microsegundos, repetidos aproximadamente cada 20 milisegundos.

Para detectar pérdida de señal se mide el tiempo transcurrido desde el último pulso válido. Si el tiempo supera un umbral definido, el sistema activa una condición de falla.



Pulso RC válido: 1000–2000 µs. Si no se detectan pulsos válidos durante más de 500 ms, el sistema entra en modo seguro.

Figura 4.4: Detección de pérdida de señal RC mediante temporización.

4.8. Diseño electrónico

El diseño electrónico del supervisor se organiza en cuatro bloques:

- Bloque de medición de voltaje.
- Bloque de medición de corriente.
- Bloque de lectura de señal RC.
- Bloque de salida de seguridad y telemetría.

Tabla 4.2: Asignación sugerida de pines para Arduino Nano.

Pin Arduino Nano	Conexión	Descripción
A0	Divisor resistivo	Medición de voltaje de batería
A1	ACS712	Medición de corriente
D2	Señal RC	Entrada con interrupción
D6	Buzzer	Alarma sonora
D8	Salida de seguridad	Habilitación/deshabilitación del robot
D10	RX SoftwareSerial	Bluetooth HC-05
D11	TX SoftwareSerial	Bluetooth HC-05
D13	LED integrado	Indicador de estado

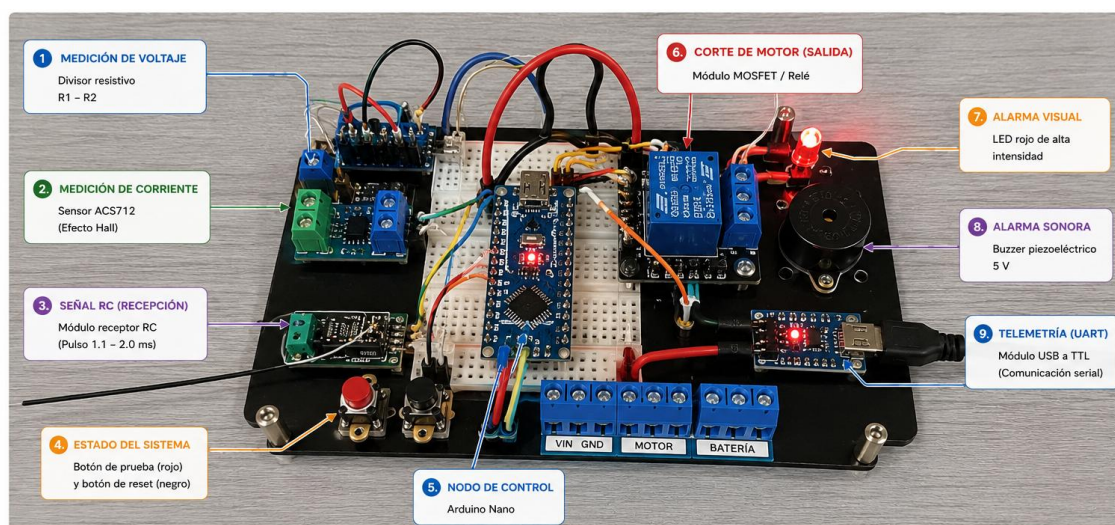


Figura 4.5: Diagrama de conexiones del supervisor inteligente de seguridad.

La salida de seguridad del Arduino no debe cortar directamente corrientes elevadas. Debe utilizarse para habilitar un driver, controlar un relé, activar un MOSFET o enviar una señal lógica al controlador principal.

4.9. Organización modular del proyecto

El proyecto puede organizarse en bloques funcionales para facilitar su desarrollo, prueba y mantenimiento:

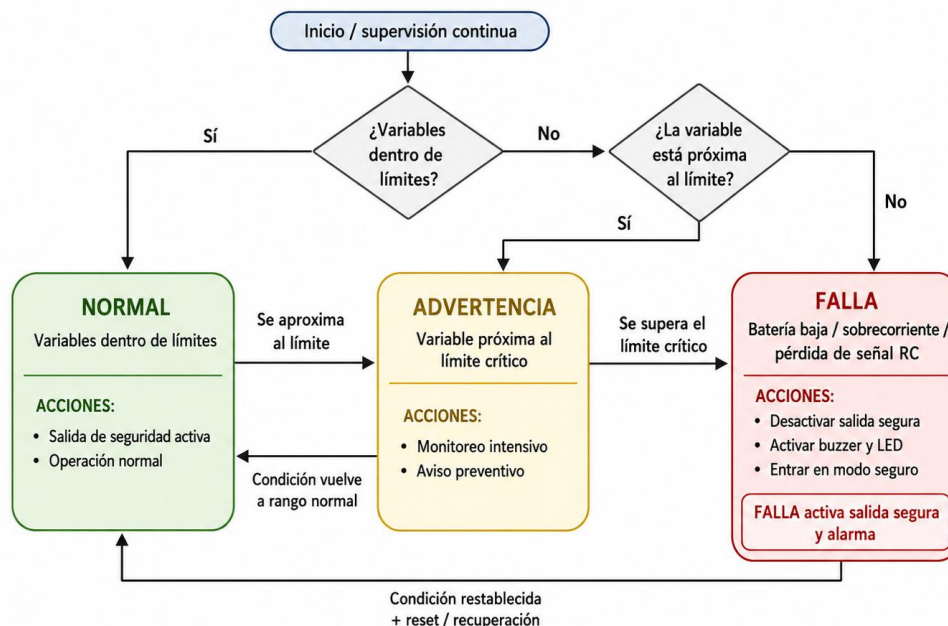
Tabla 4.3: Bloques funcionales utilizados en el supervisor inteligente.

Bloque funcional	Aplicación en el proyecto
Núcleo	Temporización no bloqueante y ciclo principal
Sensores	Lectura de voltaje, corriente y señal RC
Seguridad	Evaluación de fallas y modo seguro
Telemetría	Envío de datos por serial o Bluetooth
Actuadores	Activación de buzzer, LED y salida de seguridad
Almacenamiento	Almacenamiento Registro de eventos en EEPROM

4.10. Lógica de funcionamiento

El sistema opera en tres estados principales:

- **NORMAL:** todas las variables se encuentran dentro de los límites permitidos.
- **ADVERTENCIA:** una variable se aproxima a un límite crítico.
- **FALLA:** se detecta batería baja, sobrecorriente o pérdida de señal.



Nota: Estados principales: NORMAL, ADVERTENCIA y FALLA.

Figura 4.6: Máquina de estados del supervisor inteligente de seguridad.

4.11. Código del proyecto

El siguiente código implementa una versión funcional del supervisor inteligente utilizando Arduino Nano. La salida de seguridad se mantiene activa mientras las condiciones son normales y se desactiva ante una condición crítica [2, 8].

```
#include <SoftwareSerial.h>

// =====
// CONFIGURACION DE PINES
// =====

const byte PIN_VBAT = A0;
const byte PIN_CORRIENTE = A1;
const byte PIN_RC = 2;
const byte PIN_BUZZER = 6;
const byte PIN_SEGURIDAD = 8;
const byte PIN_LED = 13;

const byte PIN_BT_RX = 10;
const byte PIN_BT_TX = 11;

SoftwareSerial BT(PIN_BT_RX, PIN_BT_TX);

// =====
// PARAMETROS DE MEDICION
// =====

const float VREF = 5.0;
const float ADC_MAX = 1023.0;
// Divisor resistivo: R1 arriba, R2 abajo
const float R1 = 30000.0;
const float R2 = 10000.0;
// ACS712
// Usar 0.185 para 5A, 0.100 para 20A, 0.066 para 30A
const float SENSIBILIDAD_ACS = 0.100;
float voltajeCeroACS = 2.50;

// Umbrales de seguridad
const float UMBRAL_BATERIA_BAJA = 10.5; // ejemplo para LiPo 3S
const float UMBRAL_CORRIENTE_MAX = 15.0; // ajustar segun robot
const unsigned long TIMEOUT_RC = 500; // ms sin senal
// =====
// VARIABLES DE ESTADO
// =====

volatile unsigned long tiempoInicioPulso = 0;
volatile unsigned int anchoPulsoRC = 1500;
volatile bool nuevoPulsoRC = false;
unsigned long ultimoPulsoValido = 0;
```

```

unsigned long tiempoTelemetria = 0;
float voltajeBateria = 0;
float corriente = 0;

enum EstadoSupervisor
{
    NORMAL,
    ADVERTENCIA,
    FALLA
};

EstadoSupervisor estadoActual = NORMAL;

// =====
// INTERRUPCION PARA MEDIR PULSO RC
// =====
void medirPulsoRC()
{
    if (digitalRead(PIN_RC) == HIGH)
    {
        tiempoInicioPulso = micros();
    }
    else
    {
        unsigned long duracion = micros() - tiempoInicioPulso;

        if (duracion >= 900 && duracion <= 2100)
        {
            anchoPulsoRC = duracion;
            ultimoPulsoValido = millis();
            nuevoPulsoRC = true;
        }
    }
}

// =====
// FUNCIONES DE MEDICION
// =====
float leerVoltajeBateria()
{
    int adc = analogRead(PIN_VBAT);
    float vADC = (adc * VREF) / ADC_MAX;
    float vBAT = vADC * ((R1 + R2) / R2);
    return vBAT;
}

float leerCorrienteACS()

```

```

{
    int muestras = 50;
    float suma = 0;

    for (int i = 0; i < muestras; i++)
    {
        int adc = analogRead(PIN_CORRIENTE);
        float v = (adc * VREF) / ADC_MAX;
        suma += v;
    }

    float vProm = suma / muestras;
    float iMedida = (vProm - voltajeCeroACS) / SENSIBILIDAD_ACS;
    if (iMedida < 0)
    {
        iMedida = -iMedida;
    }

    return iMedida;
}

void calibrarACS()
{
    int muestras = 200;
    float suma = 0;
    for (int i = 0; i < muestras; i++)
    {
        int adc = analogRead(PIN_CORRIENTE);
        float v = (adc * VREF) / ADC_MAX;
        suma += v;
        delay(3);
    }

    voltajeCeroACS = suma / muestras;
}

// =====
// SEGURIDAD
// =====
bool senalRCPresente()
{
    return (millis() - ultimoPulsoValido) < TIMEOUT_RC;
}

void evaluarSeguridad()
{
    bool fallaBateria = voltajeBateria < UMBRAL_BATERIA_BAJA;

```

```

bool fallaCorriente = corriente > UMBRAL_CORRIENTE_MAX;
bool fallaRC = !senalRCPresente();
if (fallaBateria || fallaCorriente || fallaRC)
{
    estadoActual = FALLA;
}
else if (voltajeBateria < (UMBRAL_BATERIA_BAJA + 0.6) ||
        corriente > (UMBRAL_CORRIENTE_MAX * 0.80))
{
    estadoActual = ADVERTENCIA;
}
else
{
    estadoActual = NORMAL;
}
}

void aplicarSalidaSeguridad()
{
    if (estadoActual == FALLA)
    {
        digitalWrite(PIN_SEGURIDAD, LOW);
        digitalWrite(PIN_LED, HIGH);
        tone(PIN_BUZZER, 2000);
        registrarFallaEEPROM();
    }
    else if (estadoActual == ADVERTENCIA)
    {
        digitalWrite(PIN_SEGURIDAD, HIGH);
        digitalWrite(PIN_LED, millis() / 250 % 2);
        noTone(PIN_BUZZER);
    }
    else
    {
        digitalWrite(PIN_SEGURIDAD, HIGH);
        digitalWrite(PIN_LED, LOW);
        noTone(PIN_BUZZER);
    }
}

// =====
// EEPROM
// =====
void registrarFallaEEPROM()
{
    static bool fallaRegistrada = false;
    if (!fallaRegistrada)

```



```

{
  byte contador = EEPROM.read(0);
  contador++;
  EEPROM.write(0, contador);
  fallaRegistrada = true;
}
}

// =====
//  TELEMETRIA
// =====

void enviarTelemetry()
{
  if (millis() - tiempoTelemetry >= 250)
  {
    tiempoTelemetry = millis();

    String estadoTexto = "NORMAL";
    if (estadoActual == ADVERTENCIA) estadoTexto = "ADVERTENCIA";
    if (estadoActual == FALLA) estadoTexto = "FALLA";

    Serial.print("VBAT:");
    Serial.print(voltajeBateria, 2);
    Serial.print(", I:");
    Serial.print(corriente, 2);
    Serial.print(", RC:");
    Serial.print(anchoPulsoRC);
    Serial.print(", ESTADO:");
    Serial.println(estadoTexto);
    BT.print("VBAT:");
    BT.print(voltajeBateria, 2);
    BT.print(", I:");
    BT.print(corriente, 2);
    BT.print(", RC:");
    BT.print(anchoPulsoRC);
    BT.print(", ESTADO:");
    BT.println(estadoTexto);
  }
}

// =====
//  SETUP Y LOOP
// =====

void setup()
{
  pinMode(PIN_RC, INPUT);
  pinMode(PIN_BUZZER, OUTPUT);
}

```

```

pinMode(PIN_SEGURIDAD, OUTPUT);
pinMode(PIN_LED, OUTPUT);
digitalWrite(PIN_SEGURIDAD, LOW);
Serial.begin(115200);
BT.begin(9600);

calibrarACS();

ultimoPulsoValido = millis();
attachInterrupt(digitalPinToInterrupt(PIN_RC), medirPulsoRC, CHANGE);
Serial.println("Supervisor inteligente iniciado");
}

void loop()
{
    voltajeBateria = leerVoltajeBateria();
    corriente = leerCorrienteACS();

    evaluarSeguridad();
    aplicarSalidaSeguridad();
    enviarTelemetria();
}

```

Los valores de umbral incluidos en el código son referenciales. Deben ajustarse de acuerdo con el número de celdas de la batería, el consumo real de los motores y las especificaciones del driver o ESC utilizado.

4.12. Pruebas experimentales

Antes de instalar el sistema en un robot real, se recomienda realizar pruebas por etapas.

4.12.1. Prueba 1: medición de voltaje

Conectar una fuente o batería conocida y comparar el valor mostrado por el supervisor con la medición de un multímetro.

Tabla 4.4: Formato sugerido para validación de medición de voltaje.

Prueba	Multímetro	Arduino	Error
1			
2			
3			

4.12.2. Prueba 2: medición de corriente

Conectar una carga controlada y comparar la corriente estimada por el ACS712 con un multímetro en modo amperímetro o una pinza amperimétrica adecuada.

4.12.3. Prueba 3: pérdida de señal RC

Desconectar o apagar el transmisor RC y verificar que el sistema cambie al estado de FALLA en un tiempo menor al umbral definido.

4.12.4. Prueba 4: activación de salida segura

Conectar la salida de seguridad a un LED o entrada de habilitación del driver y verificar que se desactive en caso de falla.

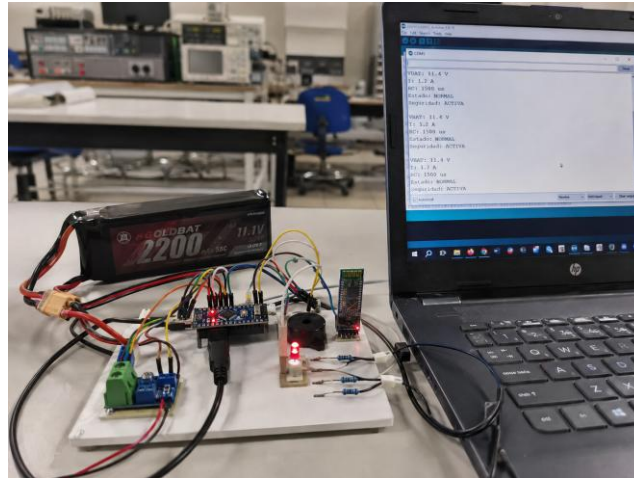


Figura 4.7: Validación experimental del supervisor inteligente de seguridad.

4.13. Extensión con pantalla OLED

Una mejora opcional consiste en agregar una pantalla OLED para visualizar datos directamente en el robot. Esta pantalla puede mostrar voltaje, corriente, estado del sistema y número de fallas registradas [6].

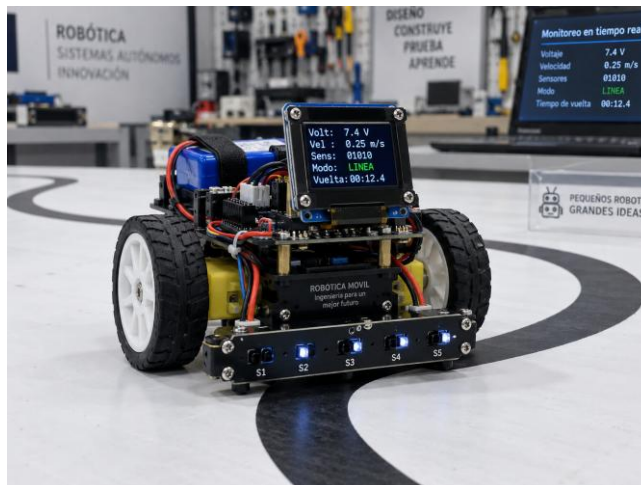


Figura 4.8: Visualización local de telemetría mediante pantalla OLED.

4.14. Extensión con App Inventor

La comunicación Bluetooth permite desarrollar una aplicación móvil sencilla en App Inventor. La aplicación puede mostrar datos de telemetría en tiempo real y alertar al usuario cuando el robot se encuentra en estado de advertencia o falla.

Tabla 4.5: Elementos sugeridos para una interfaz móvil de telemetría.

Elemento de interfaz	Función
Etiqueta de voltaje	Mostrar voltaje de batería
Etiqueta de corriente	Mostrar consumo instantáneo
Etiqueta de estado	Mostrar NORMAL, ADVERTENCIA o FALLA
Indicador visual	Cambiar color según estado
Botón conectar	Establecer conexión Bluetooth



Figura 4.9: Aplicación móvil para visualización de telemetría del supervisor inteligente.

4.15. Análisis de resultados

Después de realizar las pruebas, el estudiante debe analizar:

- Diferencia entre voltaje medido y voltaje real.
- Estabilidad de la lectura de corriente.

- Tiempo de respuesta ante pérdida de señal.
- Comportamiento de la salida de seguridad.
- Efecto del ruido eléctrico durante el movimiento del robot.

Una forma útil de analizar los datos consiste en graficar voltaje y corriente durante una prueba de aceleración. Esta información permite identificar caídas de voltaje, picos de consumo y posibles condiciones de sobrecarga.



Figura 4.10: Análisis de telemetría del supervisor durante una prueba dinámica.

4.16. Errores frecuentes

Durante el montaje del proyecto pueden aparecer los siguientes errores:

- Conectar el divisor resistivo con valores que superan el voltaje máximo del ADC.
- No unir las referencias de tierra necesarias entre módulos.
- Conectar el ACS712 en sentido incorrecto.
- Utilizar un umbral de batería baja no compatible con el número de celdas.
- Pretender cortar corriente de motores directamente desde un pin Arduino.
- No calibrar el valor de cero del ACS712.
- Usar cables largos sin considerar ruido eléctrico.

4.17. Proyecto de extensión

Como actividad de ampliación, se propone desarrollar una versión avanzada del supervisor utilizando ESP32. Esta versión podría incorporar:

- Dashboard web.
- Registro de datos en memoria Flash.
- Comunicación WiFi.
- Configuración remota de umbrales.
- Visualización de gráficas en tiempo real.
- Integración con MQTT.

Esta extensión servirá como base para proyectos posteriores relacionados con telemetría avanzada y PID Web Tuning.

Actividades de aprendizaje

1. Calcule un divisor resistivo adecuado para medir una batería LiPo 4S con Arduino Nano.
2. Modifique el código para trabajar con una batería 2S.
3. Cambie el umbral de corriente máxima de acuerdo con el consumo de un motor real.
4. Agregue una salida visual con tres LEDs: verde, amarillo y rojo.
5. Diseñe una app en App Inventor que muestre voltaje, corriente y estado del supervisor.
6. Registre datos de telemetría durante una prueba de movimiento y gráfíquelos en MATLAB o una hoja de cálculo.

Preguntas de reflexión

1. ¿Por qué un robot competitivo debe incorporar mecanismos de seguridad independientes del algoritmo principal?
2. ¿Qué riesgos existen al operar una batería LiPo por debajo del voltaje recomendado?
3. ¿Por qué es importante medir corriente en un robot con motores de alta potencia?
4. ¿Qué diferencia existe entre advertencia y falla dentro de una máquina de estados de seguridad?
5. ¿Cómo podría adaptarse este proyecto para un robot de combate?
6. ¿Qué ventajas tendría implementar este supervisor en ESP32?

Referencias

[1] Arduino, “Nano”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/hardware/nano>. [Accedido: 24-jun-2026].

[2] Arduino, “Arduino Language Reference”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/language-reference/>. [Accedido: 24-jun2026].

- [3] Arduino, “A guide to EEPROM”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/learn/programming/eprom-guide>. [Accedido: 24-jun2026].
- [4] Allegro MicroSystems, “ACS712 Galvanically Isolated Linear Current Sensor IC”, Allegro MicroSystems. [En línea]. Disponible en: <https://www.allegromicro.com/en/products/sense/current-sensor-ics/integrated-current-sensors/acs712>. [Accedido: 24-jun-2026].
- [5] Allegro MicroSystems, “ACS712 Datasheet”, Allegro MicroSystems. [En línea]. Disponible en: <https://www.allegromicro.com/-/media/files/datasheets/acs712-datasheet.ashx>. [Accedido: 24-jun-2026].
- [6] Adafruit, “Adafruit SSD1306 Arduino Library”, GitHub. [En línea]. Disponible en: https://github.com/adafruit/Adafruit_SSD1306. [Accedido: 24-jun-2026].
- [7] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2017.
- [8] M. Barr and A. Massa, *Programming Embedded Systems: With C and GNU Development Tools*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2006.
- [9] D. Cook, *Robot Building for Beginners*, 4th ed. Berkeley, CA, USA: Apress, 2020.

Capítulo 5. Telemetría de Impactos con MPU6050 y EEPROM

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender la importancia de la telemetría en robots competitivos sometidos a impactos, vibraciones y cambios bruscos de movimiento.
- Utilizar un sensor inercial MPU6050 para medir aceleración y velocidad angular.
- Calcular la magnitud resultante de aceleración para identificar eventos de impacto.
- Registrar eventos críticos en memoria EEPROM para análisis posterior.
- Enviar datos de aceleración por comunicación serial para graficarlos en una computadora.
- Diseñar un protocolo básico de pruebas experimentales para evaluar impactos en robots Minisumo, robots de combate o plataformas móviles.

5.1. Introducción

La mayoría de robots competitivos son evaluados principalmente por su desempeño observable: velocidad, capacidad de empuje, precisión, estabilidad o resistencia. Sin embargo, durante una competencia ocurren fenómenos internos que no siempre pueden apreciarse a simple vista. Un robot puede recibir impactos, experimentar vibraciones, perder adherencia, frenar bruscamente o sufrir aceleraciones que afectan su comportamiento y la integridad de sus componentes [6].

La telemetría permite medir y registrar información del robot durante su funcionamiento. En lugar de depender únicamente de la observación visual, el diseñador puede analizar datos reales obtenidos durante pruebas o combates. Esta información resulta especialmente útil para mejorar diseños mecánicos, identificar zonas vulnerables, evaluar la rigidez de un chasis, comparar materiales o ajustar estrategias de control [7].

En robots de combate, la medición de impactos puede ayudar a estimar la severidad de los golpes recibidos. En Minisumo, puede utilizarse para detectar empujes bruscos, choques laterales o pérdida de estabilidad. En seguidores de línea, la telemetría inercial puede evidenciar vibraciones excesivas o movimientos violentos al ingresar a curvas cerradas.

Este capítulo desarrolla un sistema de telemetría de impactos basado en Arduino Nano, sensor MPU6050 y memoria EEPROM. El objetivo es construir un módulo capaz de medir aceleraciones en tres ejes, calcular una magnitud resultante, detectar impactos que superen un umbral y registrar eventos críticos para análisis posterior.

El proyecto se plantea como una herramienta de instrumentación aplicada a robótica competitiva. No busca reemplazar instrumentos profesionales de medición, sino proporcionar

una solución educativa, económica y funcional que permita introducir al lector en el análisis experimental de datos inerciales.



Figura 5.1: Robot competitivo instrumentado con sensor inercial para registro de impactos.

5.2. Planteamiento del problema

Durante las pruebas de un robot competitivo, muchas decisiones de diseño se toman de manera intuitiva. Por ejemplo, si una pieza se afloja después de varios combates, se refuerza; si un sensor se descalibra después de un golpe, se reubica; si una batería se desplaza, se mejora el sistema de sujeción. Aunque estas acciones son válidas, resulta conveniente complementarlas con datos medibles.

Algunos problemas que pueden analizarse mediante telemetría de impactos son:

- Identificación de golpes fuertes durante combate.
- Comparación de vibraciones entre diferentes diseños de chasis.
- Detección de impactos laterales, frontales o verticales.
- Evaluación de soportes para sensores y baterías.
- Registro de eventos críticos, aunque el robot se apague posteriormente.
- Análisis de aceleraciones durante arranques, frenados y choques.

La telemetría transforma una prueba subjetiva en un experimento medible. Un robot que registra sus propios eventos permite comprender mejor qué ocurre durante el combate y facilita la mejora del diseño.

5.3. Descripción general del proyecto

El sistema propuesto utiliza un Arduino Nano conectado a un sensor MPU6050 mediante comunicación I2C. El microcontrolador lee los valores de aceleración en los ejes X, Y y Z, convierte los datos a unidades de gravedad g, calcula la aceleración resultante y compara el valor obtenido con un umbral de impacto [5].

Cuando la aceleración supera el umbral establecido, el sistema registra el evento en memoria EEPROM. Además, envía los datos por comunicación serial para visualizarlos en el monitor serial o graficarlos en una computadora.

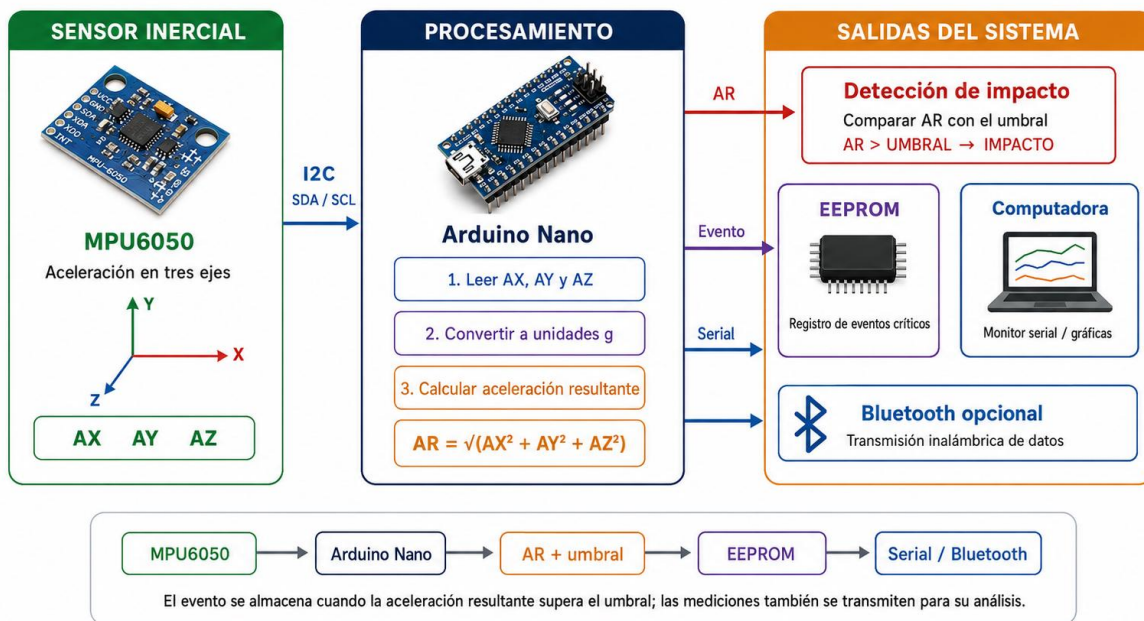


Figura 5.2: Arquitectura general del sistema de telemetría de impactos.

5.4. Aplicación STEM

Este proyecto integra los componentes de la metodología STEM de la siguiente forma:

- **Ciencia:** estudio de aceleración, vibración, fuerzas de impacto y dinámica del movimiento.
- **Tecnología:** uso de sensores MEMS, comunicación I2C, microcontroladores y memoria no volátil.
- **Ingeniería:** diseño de un sistema de medición aplicable a robots reales de competencia. Matemáticas: cálculo de magnitud vectorial, umbrales, promedios, escalamiento y análisis de datos.

5.5. Objetivos del proyecto

5.5.1. Objetivo general

Diseñar e implementar un sistema embebido de telemetría de impactos para robots competitivos, utilizando un sensor MPU6050, Arduino Nano y memoria EEPROM.

5.5.2. Objetivos específicos

1. Configurar la comunicación I2C entre Arduino Nano y MPU6050.
2. Leer datos de aceleración en los ejes X, Y y Z.
3. Convertir las lecturas crudas del sensor a unidades de gravedad g.
4. Calcular la magnitud resultante de aceleración.
5. Detectar impactos mediante un umbral configurable.
6. Registrar eventos críticos en EEPROM.
7. Enviar datos por puerto serial para análisis experimental.

5.6. Materiales y herramientas

Tabla 5.1: Materiales principales para el sistema de telemetría de impactos.

Elemento	Cantidad	Función
Arduino Nano	1	Microcontrolador
MPU6050	1	Medición de aceleración y giro
Protoboard o PCB	1	Montaje inicial o definitivo
Cables Dupont	Varios	Interconexión
Batería o fuente de 5 V	1	Alimentación del sistema
Computadora con Arduino IDE	1	Programación y monitoreo serial
Robot de prueba	1	Plataforma de validación experimental

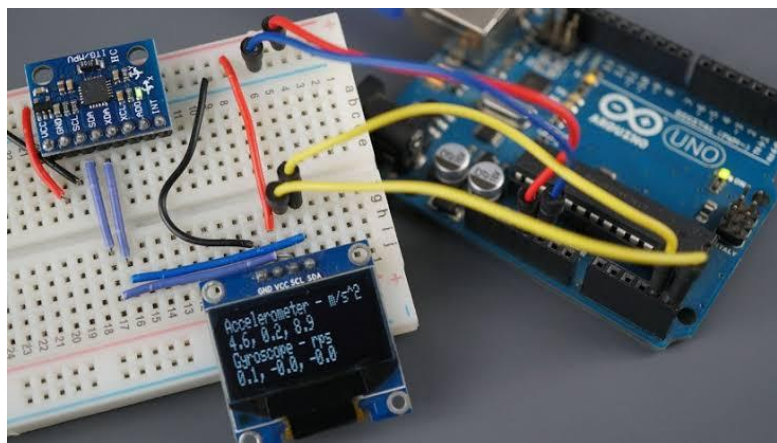


Figura 5.3: Materiales utilizados para construir el sistema de telemetría de impactos.

5.7. Fundamento teórico

5.7.1. Sensor MPU6050

El MPU6050 es una unidad de medición inercial que integra un acelerómetro de tres ejes y un giroscopio de tres ejes. La documentación técnica del dispositivo describe al MPU-60X0 como un sistema que combina giroscopio MEMS de tres ejes, acelerómetro MEMS de tres ejes y un procesador digital de movimiento [4]. En este proyecto se utilizará principalmente el acelerómetro [1, 2, 3].

El módulo se comunica con Arduino mediante el protocolo I2C, utilizando normalmente las líneas SDA y SCL. En Arduino Nano, estas líneas corresponden a los pines A4 y A5.

Tabla 5.2: Conexiones básicas entre Arduino Nano y MPU6050.

MPU6050	Arduino Nano	Descripción
VCC	5 V o 3.3 V según módulo	Alimentación
GND	GND	Tierra común
SDA	A4	Datos I2C
SCL	A5	Reloj I2C
INT	No conectado inicialmente	Interrupción opcional

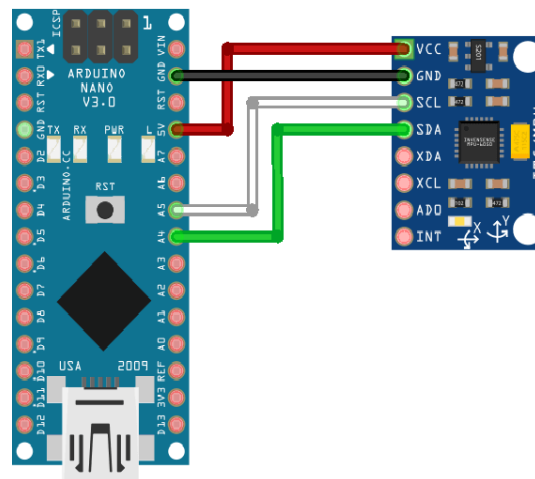


Figura 5.4: Conexión física del MPU6050 con Arduino Nano.

5.7.2. Comunicación I2C

I2C es un protocolo de comunicación serial que permite conectar múltiples dispositivos mediante dos líneas principales: SDA para datos y SCL para reloj. Arduino utiliza la librería Wire para comunicarse con dispositivos I2C [3, 4].

En muchos módulos MPU6050, la dirección I2C por defecto es:

0x68

Esta dirección puede variar si se modifica el pin AD0 del sensor.

5.7.3. Aceleración en tres ejes

El acelerómetro del MPU6050 entrega valores en tres ejes: a_x , a_y y a_z . Estos valores permiten conocer la aceleración del sistema en las direcciones longitudinal, lateral y vertical.

La aceleración resultante puede calcularse como la magnitud del vector:

$$a_R = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

Cuando el sensor está en reposo, la aceleración resultante suele ser cercana a 1_g , debido a la gravedad terrestre. Durante un impacto, esta magnitud puede aumentar rápidamente.

5.7.4. Detección de impacto

Para detectar un impacto se define un umbral:

$$a_R > a_{umbral}$$

Si la aceleración resultante supera el umbral, el sistema registra un evento.

El umbral debe ajustarse experimentalmente. Un valor muy bajo generará falsas detecciones durante movimientos normales. Un valor demasiado alto podría ignorar impactos relevantes.

Tabla 5.3: Ejemplo referencial de interpretación de aceleración resultante.

Magnitud aproximada	Interpretación posible
1 g	Reposo o movimiento suave
1.5 g – 2.5 g	Aceleración o frenado moderado
2.5 g – 4 g	Golpe o cambio brusco de movimiento
Mayor a 4 g	Impacto fuerte o vibración severa

Los rangos de interpretación dependen del robot, la ubicación del sensor, la rigidez del chasis y el tipo de impacto. Por ello, los valores deben considerarse referenciales y deben calibrarse mediante pruebas reales.

5.7.5. Registro en EEPROM

La EEPROM permite conservar datos incluso cuando el microcontrolador se apaga. Arduino dispone de una librería EEPROM para leer y escribir valores no volátiles [3]. En este proyecto se utilizará para almacenar la cantidad de impactos detectados y el mayor valor de aceleración registrado.

Debe considerarse que la EEPROM posee un número limitado de ciclos de escritura. Por ello, no se debe escribir continuamente en cada lectura del sensor, sino únicamente cuando ocurre un evento importante.

5.8. Diseño electrónico

El circuito es relativamente simple. El MPU6050 se conecta al bus I2C del Arduino Nano. El sistema puede alimentarse desde el mismo regulador del robot o desde una batería auxiliar durante pruebas.



Figura 5.5: Conexiones del sistema de telemetría de impactos.

Tabla 5.4: Asignación sugerida de pines.

Pin Arduino Nano	Conexión	Función
A4	SDA MPU6050	Comunicación I2C
A5	SCL MPU6050	Comunicación I2C
5V	VCC MPU6050	Alimentación según módulo
GND	GND MPU6050	Tierra común
D13	LED indicador	Evento de impacto
USB	Computadora	Telemetría serial

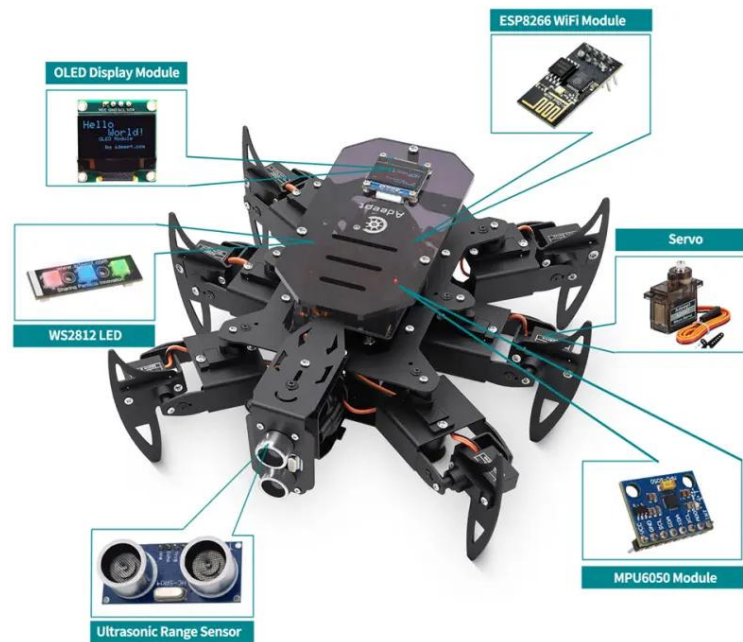


Figura 5.6: Montaje del sistema de telemetría de impactos en el robot.

5.9. Organización modular del programa

El software se organiza en los siguientes bloques:

- Inicialización del MPU6050.
- Lectura de registros de aceleración.
- Conversión de datos crudos a unidades g.
- Cálculo de aceleración resultante.
- Detección de impactos.
- Registro de eventos en EEPROM.
- Envío de telemetría por puerto serial.

Esta organización facilita modificar el proyecto posteriormente para agregar pantalla OLED, Bluetooth, tarjeta SD o ESP32.

5.10. Código del proyecto

El siguiente programa lee directamente los registros del MPU6050 utilizando la librería Wire. Se configura el sensor en rango de aceleración de $\pm 2g$, cuya sensibilidad es de 16384 LSB/g según la configuración estándar del dispositivo.

```
#include <Wire.h>
#include <EEPROM.h>
// =====
// CONFIGURACION DEL MPU6050
```

```

// =====
const byte MPU_ADDR = 0x68;

// Registros principales
const byte REG_PWR_MGMT_1 = 0x6B;
const byte REG_ACCEL_XOUT_H = 0x3B;

// Sensibilidad para rango +-2g
const float SENSIBILIDAD_ACCEL = 16384.0;

// =====
// PARAMETROS DEL PROYECTO
// =====
const byte PIN_LED = 13;
const float UMBRAL_IMPACTO_G = 3.0;
const unsigned long TIEMPO_ENTRE_EVENTOS = 500;

// Direcciones EEPROM 24 const int EEPROM_CONTADOR = 0;
const int EEPROM_MAX_G = 4;

// =====
// VARIABLES GLOBALES
// =====
float ax, ay, az;
float aceleracionResultante = 0;
float maximoRegistrado = 0;

unsigned long ultimoImpacto = 0;
unsigned long tiempoTelemetria = 0;

// =====
// FUNCIONES EEPROM PARA FLOAT
// =====
void escribirFloatEEPROM(int direccion, float valor)
{
    EEPROM.put(direccion, valor);
}

float leerFloatEEPROM(int direccion)
{
    float valor;
    EEPROM.get(direccion, valor);

    if (isnan(valor) || valor < 0 || valor > 100)
    {
        valor = 0;
    }
}

```



```

    return valor;
}

// =====
// INICIALIZACION DEL MPU6050
// =====
void iniciarMPU6050()
{
    Wire.begin();

    Wire.beginTransmission(MPU_ADDR);
    Wire.write(REG_PWR_MGMT_1);
    Wire.write(0);
    Wire.endTransmission(true);
}

// =====
// LECTURA DE ACELERACION
// =====
int16_t leerWordMPU(byte registro)
{
    Wire.beginTransmission(MPU_ADDR);
    Wire.write(registro);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU_ADDR, (byte)2, true);

    int16_t valor = Wire.read() << 8 | Wire.read();
    return valor;
}

void leerAceleracion()
{
    int16_t rawAx = leerWordMPU(REG_ACCEL_XOUT_H);
    int16_t rawAy = leerWordMPU(REG_ACCEL_XOUT_H + 2);
    int16_t rawAz = leerWordMPU(REG_ACCEL_XOUT_H + 4);

    ax = rawAx / SENSIBILIDAD_ACCEL;
    ay = rawAy / SENSIBILIDAD_ACCEL;
    az = rawAz / SENSIBILIDAD_ACCEL;

    aceleracionResultante = sqrt(ax * ax + ay * ay + az * az);
}

// =====
// DETECCION Y REGISTRO DE IMPACTOS
// =====

```

```

void registrarImpacto()
{
    byte contador = EEPROM.read(EEPROM_CONTADOR);
    contador++;
    EEPROM.write(EEPROM_CONTADOR, contador);

    if (aceleracionResultante > maximoRegistrado)
    {
        maximoRegistrado = aceleracionResultante;
        escribirFloatEEPROM(EEPROM_MAX_G, maximoRegistrado);
    }
}

void evaluarImpacto()
{
    bool superaUmbral = aceleracionResultante >= UMBRAL_IMPACTO_G;
    bool tiempoValido = millis() - ultimoImpacto >= TIEMPO_ENTRE_EVENTOS;

    if (superaUmbral && tiempoValido)
    {
        ultimoImpacto = millis();
        registrarImpacto();

        digitalWrite(PIN_LED, HIGH);

        Serial.print("IMPACTO DETECTADO, G=");
        Serial.println(aceleracionResultante, 2);
    }
    else
    {
        digitalWrite(PIN_LED, LOW);
    }
}

// =====
// TELEMETRIA
// =====
void enviarTelemetria()
{
    if (millis() - tiempoTelemetria >= 100)
    {
        tiempoTelemetria = millis();

        Serial.print("AX:");
        Serial.print(ax, 2);
        Serial.print(", AY:");
        Serial.print(ay, 2);
    }
}

```

```

    Serial.print(", AZ:");
    Serial.print(az, 2);
    Serial.print(", AR:");
    Serial.print(acceleracionResultante, 2);
    Serial.print(", IMPACTOS:");
    Serial.print(EEPROM.read(EEPROM_CONTADOR));
    Serial.print(", MAX_G:");
    Serial.println(maximoRegistrado, 2);
}
}

// =====
// COMANDOS POR SERIAL
// =====
void revisarComandos()
{
    if (Serial.available())
    {
        char c = Serial.read();

        if (c == 'r' || c == 'R')
        {
            EEPROM.write(EEPROM_CONTADOR, 0);
            escribirFloatEEPROM(EEPROM_MAX_G, 0);
            maximoRegistrado = 0;
            Serial.println("Registros reiniciados");
        }
    }
}

// =====
// SETUP Y LOOP
// =====
void setup()
{
    pinMode(PIN_LED, OUTPUT);

    Serial.begin(115200);
    iniciarMPU6050();

    maximoRegistrado = leerFloatEEPROM(EEPROM_MAX_G);

    Serial.println("Telemetria de impactos iniciada");
    Serial.println("Enviar R para reiniciar registros");
}

void loop()

```

```

{
  leerAceleracion();
  evaluarImpacto();
  enviarTelemetria();
  revisarComandos();
}

```

El programa escribe en EEPROM únicamente cuando detecta un impacto. Esto evita desgaste innecesario de la memoria. No se recomienda guardar datos en EEPROM en cada iteración del ciclo principal.

5.11. Calibración del sistema

Antes de instalar el módulo en un robot competitivo, se recomienda realizar pruebas controladas.

5.11.1. Prueba en reposo

Colocar el módulo sobre una mesa firme y observar la aceleración resultante. En reposo debería ser cercana a 1g, aunque pueden existir pequeñas variaciones por ruido o inclinación del sensor.

5.11.2. Prueba de orientación

Girar el módulo en diferentes posiciones y verificar que los ejes X, Y y Z cambien de acuerdo con la orientación del sensor.

5.11.3. Prueba de impacto suave

Golpear ligeramente la superficie donde se encuentra el sensor y observar si la aceleración resultante aumenta sin superar necesariamente el umbral definido.

5.11.4. Prueba de umbral

Ajustar el valor de UMBRAL_IMPACTO_G hasta que el sistema ignore movimientos normales, pero detecte golpes relevantes.

Tabla 5.5: Formato sugerido para calibración del umbral de impacto.

Prueba	Condición	Máximo medio	¿Detecta impacto?
1	Reposo		
2	Movimiento suave		
3	Frenado brusco		
4	Golpe frontal		
5	Golpe lateral		

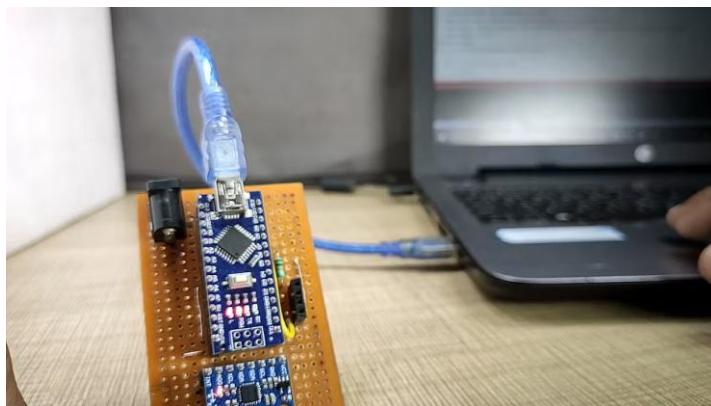


Figura 5.7: Calibración experimental del sistema de telemetría de impactos.

5.12. Pruebas experimentales en robots competitivos

El sistema puede validarse en diferentes plataformas.

5.12.1. Minisumo

Instalar el sensor cerca del centro de masa del robot. Realizar pruebas de empuje contra un obstáculo fijo y combates controlados contra otro robot.

5.12.2. Robot de combate

Fijar el módulo dentro del chasis, protegido contra golpes directos. Registrar eventos durante pruebas de impacto controladas o combates de baja intensidad.

5.12.3. Seguidor de línea

Instalar el módulo en la parte central del robot y registrar vibraciones durante curvas cerradas, frenados o cambios bruscos de dirección.



Figura 5.8: Prueba experimental de telemetría de impactos en un robot competitivo.

5.13. Análisis de datos

Los datos enviados por el puerto serial pueden copiarse y analizarse en una hoja de cálculo, MATLAB, Python o cualquier herramienta de graficación. Un análisis básico consiste en graficar la aceleración resultante en función del tiempo.

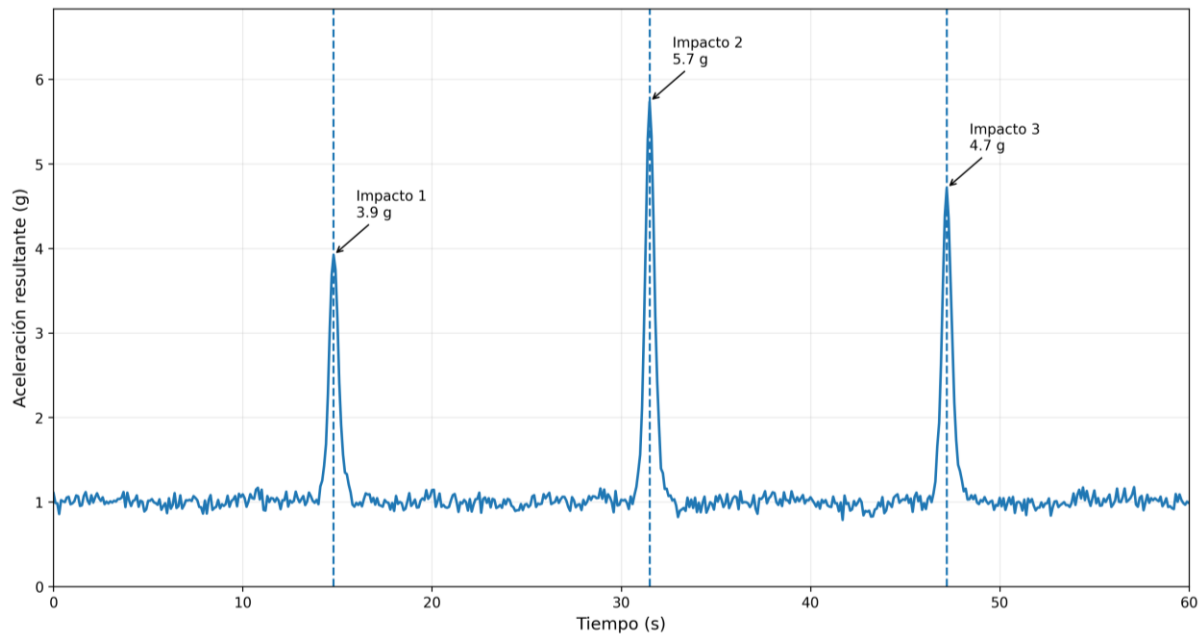


Figura 5.9: Gráfica de aceleración resultante durante una prueba de impacto.

La interpretación de los datos debe considerar:

- Magnitud máxima registrada.
- Frecuencia de impactos.
- Dirección predominante del impacto.
- Diferencia entre vibración normal e impacto crítico.
- Relación entre eventos de impacto y fallas mecánicas observadas.

5.14. Visualización en MATLAB

Como extensión, los datos seriales pueden enviarse a MATLAB para graficar la aceleración resultante en tiempo real. El siguiente pseudocódigo representa la lógica básica:

```
Abrir puerto serial
Mientras el sistema este activo:
  Leer linea del puerto serial
  Extraer AX, AY, AZ y AR
  Guardar datos en vectores
  Graficar AR en funcion del tiempo
Cerrar puerto serial
```

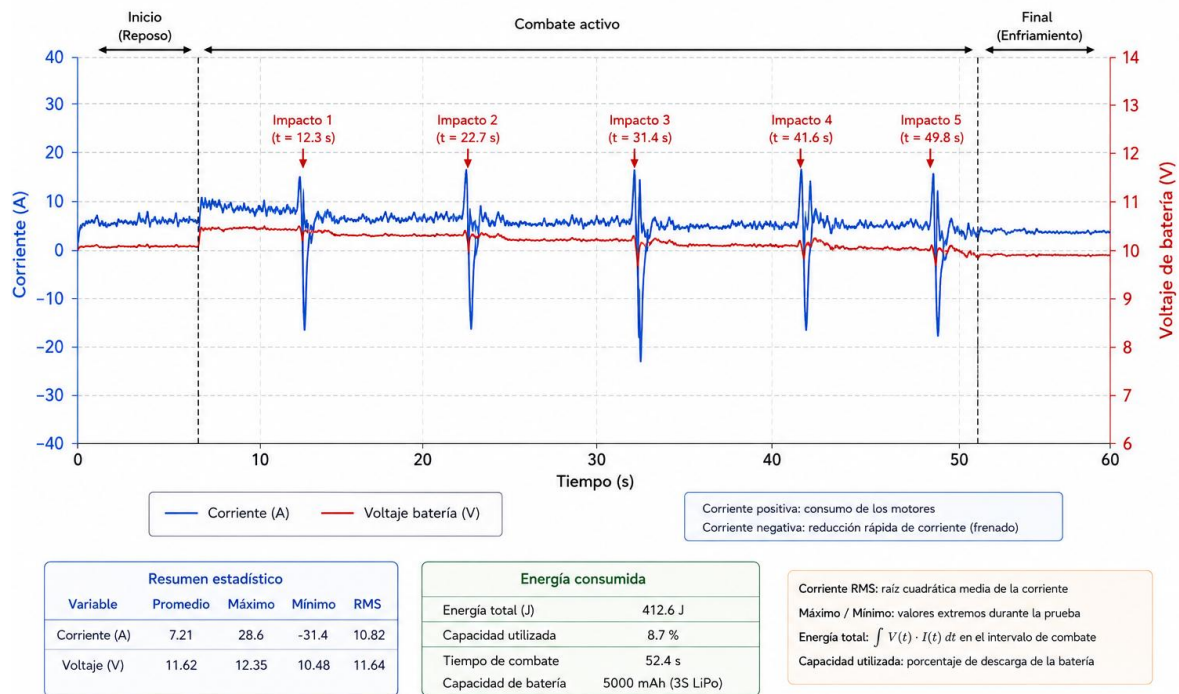


Figura 5.10: Visualización de datos de impacto en una herramienta de análisis.

5.15. Errores frecuentes

Durante el desarrollo del proyecto pueden presentarse los siguientes errores:

- Conectar SDA y SCL en pines incorrectos.
- Alimentar el módulo MPU6050 con un voltaje no compatible con su placa.
- No fijar correctamente el sensor al chasis, generando lecturas erráticas.
- Definir un umbral demasiado bajo y detectar falsos impactos.
- Escribir en EEPROM con demasiada frecuencia.
- Interpretar vibraciones normales del motor como impactos críticos.
- Instalar el sensor en una zona flexible del robot.

5.16. Proyecto de extensión

Se propone desarrollar una versión avanzada del sistema utilizando ESP32. Esta versión podría incorporar:

- Visualización web en tiempo real.
- Registro de datos en tarjeta microSD.
- Configuración inalámbrica del umbral de impacto.
- Envío de alertas a una aplicación móvil.
- Análisis comparativo entre diferentes chasis.

Otra extensión consiste en combinar la telemetría de impactos con medición de corriente. Esto permitiría relacionar golpes, bloqueos de motor y picos de consumo durante un combate.

Actividades de aprendizaje

1. Monte el MPU6050 sobre una protoboard y verifique su lectura en reposo.
2. Registre valores de aceleración durante movimientos suaves y golpes controlados.
3. Ajuste el umbral de impacto para evitar falsas detecciones.
4. Instale el módulo en un robot Minisumo y registre impactos durante pruebas de empuje.
5. Grafique la aceleración resultante en una hoja de cálculo o MATLAB.
6. Modifique el programa para guardar los tres mayores impactos detectados.

Preguntas de reflexión

1. ¿Por qué la aceleración resultante en reposo suele ser cercana a 1g?
2. ¿Qué ventajas ofrece registrar eventos de impacto en EEPROM?
3. ¿Por qué no es recomendable escribir datos en EEPROM continuamente?
4. ¿Cómo influye la ubicación del sensor en las mediciones obtenidas?
5. ¿Qué diferencias esperaría encontrar entre impactos frontales y laterales?
6. ¿Cómo podría utilizarse esta información para mejorar el diseño mecánico de un robot?

Referencias

- [1] InvenSense, “MPU-6000 and MPU-6050 Product Specification”, Rev. 3.4, 2013. [En línea]. Disponible en: <https://cdn.sparkfun.com/datasheets/Components/General%20IC/PS-MPU-6000A.pdf>. [Accedido: 24-jun-2026].
- [2] TDK InvenSense, “MPU-6000 and MPU-6050 Register Map and Descriptions”, Rev. 4.2. [En línea]. Disponible en: <https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf>. [Accedido: 24-jun-2026].
- [3] TDK Product Center, “MPU-6050: Detailed Information”. [En línea]. Disponible en: https://product.tdk.com/en/search/sensor/motion-inertial/imu/info?part_no=MPU-6050. [Accedido: 24-jun-2026].
- [4] Arduino, “Wire Library”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/learn/communication/wire>. [Accedido: 24-jun-2026].
- [5] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2017.
- [6] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed. Cham, Switzerland: Springer, 2016.
- [7] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.

Capítulo 6. Control Seguro de Motores Brushless con ESC

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender el principio de control de un motor brushless mediante un controlador electrónico de velocidad ESC.
- Generar señales de control tipo servo mediante Arduino para gobernar un ESC.
- Implementar una rampa de arranque progresiva para reducir cambios bruscos de corriente y velocidad.
- Incorporar una lógica básica de armado, desarmado y parada de emergencia.
- Diseñar una prueba segura de banco para motores brushless usados en sistemas auxiliares de robots competitivos.
- Registrar datos básicos de operación mediante monitor serial.

6.1. Introducción

Los motores brushless, también conocidos como motores BLDC, son ampliamente utilizados en aplicaciones donde se requiere alta velocidad, buena eficiencia y elevada relación potencia-peso. Se encuentran en drones, ventiladores de alto rendimiento, turbinas, sistemas de succión, plataformas móviles, aeromodelismo y algunos sistemas auxiliares de robots competitivos.

A diferencia de un motor DC convencional, un motor brushless no puede conectarse directamente a una fuente de alimentación ni a un puente H simple. Requiere un controlador electrónico de velocidad, conocido como ESC (*Electronic Speed Controller*), encargado de conmutar electrónicamente las fases del motor de acuerdo con la señal de control recibida.

En robótica competitiva, los motores brushless deben tratarse con especial cuidado. Un arranque brusco puede generar picos de corriente, vibraciones, pérdida de control, sobrecalentamiento o daños mecánicos. Por esta razón, es recomendable implementar secuencias de armado, rampas de aceleración, límites de señal y mecanismos de parada de emergencia.

Este capítulo desarrolla un sistema de control seguro para motores brushless mediante Arduino Nano y ESC. El enfoque del proyecto se orienta a sistemas auxiliares de robots competitivos, como ventiladores, turbinas de succión, bancos de prueba, sistemas de enfriamiento o demostradores didácticos. El capítulo no aborda el diseño de mecanismos de impacto, corte o contacto; su propósito es enseñar control, supervisión y operación segura del conjunto motor-ESC.

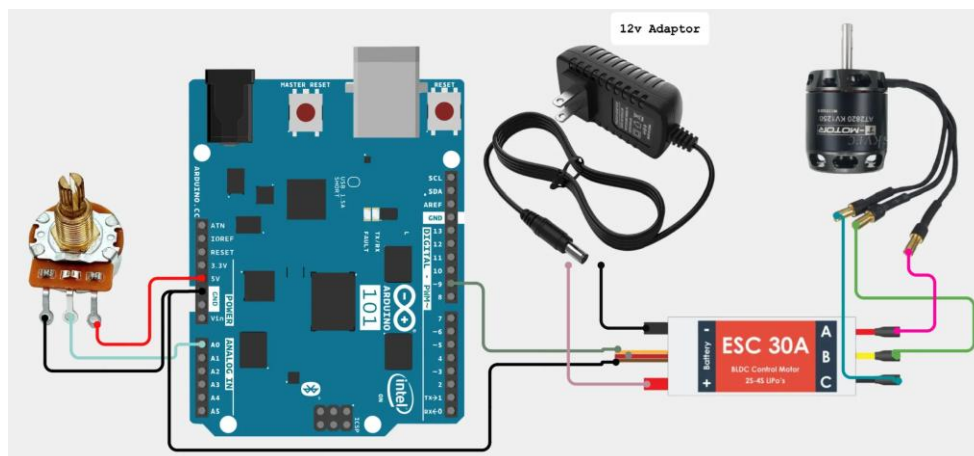


Figura 6.1: Sistema básico para control seguro de un motor brushless mediante ESC.

6.2. Planteamiento del problema

Un motor brushless puede alcanzar velocidades elevadas en muy poco tiempo. Si el sistema se activa de forma accidental, si la señal del ESC no está correctamente inicializada o si se aplica un cambio brusco de consigna, el montaje puede comportarse de manera peligrosa o dañar componentes electrónicos.

Entre los problemas más comunes se encuentran:

- Activación accidental del motor al energizar el sistema.
- Picos de corriente durante el arranque.
- Calentamiento del ESC o del motor.
- Pérdida de sincronismo por aceleración excesiva.
- Señales PWM fuera del rango esperado por el ESC.
- Ausencia de parada de emergencia.
- Pruebas realizadas sin fijación mecánica adecuada.

El proyecto propone una solución basada en tres ideas principales: armado seguro, rampa de aceleración y parada de emergencia.

Un sistema brushless no debe arrancar directamente a máxima velocidad. La señal de control debe generarse de forma progresiva, verificable y con mecanismos claros de desactivación.

6.3. Descripción general del proyecto

El sistema está compuesto por un Arduino Nano, un ESC, un motor brushless, una batería LiPo, un potenciómetro para ajustar la consigna de velocidad, un pulsador de armado, un pulsador de emergencia y un LED de estado [1].

El Arduino genera una señal de control para el ESC utilizando la librería Servo. Esta señal se expresa en microsegundos, de forma similar a la señal utilizada para servomotores RC. La documentación de Arduino indica que la librería Servo permite controlar servomotores y generar señales compatibles con este tipo de control [3]. En muchos ESCs PWM se utilizan rangos nominales cercanos a 1000 – 2000 μ s, aunque el rango exacto debe verificarse en la documentación del ESC empleado [2, 3, 4].

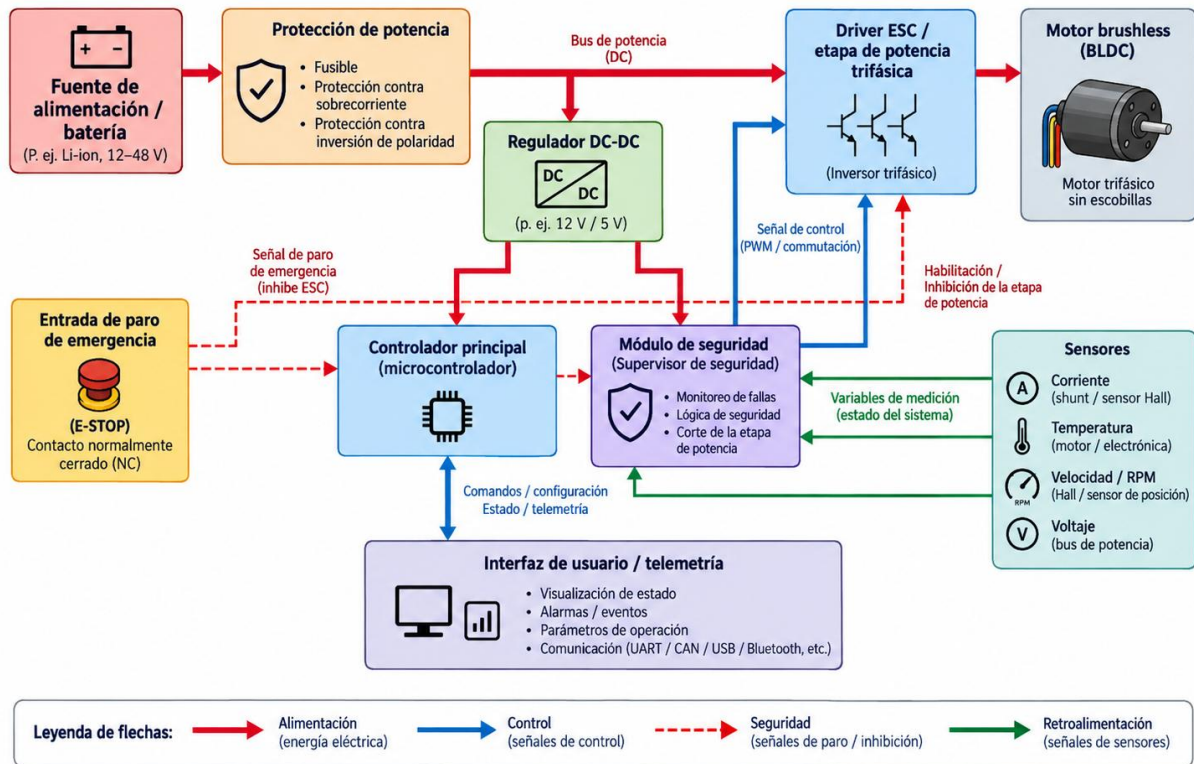


Figura 6.2: Arquitectura general del sistema de control seguro para motor brushless.

6.4. Aplicación STEM

Este proyecto integra los siguientes elementos STEM:

- **Ciencia:** análisis de velocidad, aceleración, corriente, energía y calentamiento.
- **Tecnología:** uso de ESCs, motores brushless, microcontroladores y señales PWM tipo servo.
- **Ingeniería:** diseño de una secuencia de arranque segura y control progresivo de velocidad.
- **Matemáticas:** conversión de señales analógicas, interpolación de rangos, cálculo de rampas y tiempos de aceleración.

6.5. Objetivos del proyecto

6.5.1. Objetivo general

Diseñar e implementar un sistema de control seguro para motores brushless mediante Arduino Nano y ESC, incorporando armado, rampa de arranque, límite de señal y parada de emergencia.

6.5.2. Objetivos específicos

1. Generar una señal compatible con ESC mediante la librería Servo.
2. Leer una consigna de velocidad mediante potenciómetro.
3. Implementar un estado de desarmado que mantenga el motor detenido al iniciar.
4. Implementar una rampa progresiva desde la señal mínima hasta la consigna deseada.
5. Incorporar una entrada de emergencia para detener el sistema.
6. Enviar datos de estado por el monitor serial.

6.6. Materiales y herramientas

Tabla 6.1: Materiales principales para el sistema de control brushless.

Elemento	Cantidad	Función
Arduino Nano	1	Generación de señal de control
ESC compatible con el motor	1	Control electrónico del motor brushless
Motor brushless	1	Actuador principal del banco de prueba
Batería LiPo adecuada	1	Alimentación del ESC
Potenciómetro 10 K Ω	1	Consigna de velocidad
Pulsador de armado	1	Habilitación intencional del sistema
Pulsador de emergencia	1	Desactivación inmediata
LED con resistencia	1	Indicación de estado
Cables y conectores adecuados	Varios	Interconexión eléctrica
Base de prueba rígida	1	Fijación segura del motor



Figura 6.3: Materiales utilizados para el control seguro de motores brushless.

6.7. Consideraciones de seguridad

Los motores brushless pueden ser peligrosos si se prueban sin protección o con elementos giratorios mal fijados. Por ello, este capítulo debe desarrollarse en un entorno controlado y bajo supervisión docente.

Recomendaciones mínimas:

- No montar hélices, discos, cuchillas ni elementos de impacto durante las primeras pruebas.
- Fijar el motor a una base rígida antes de energizar el ESC.
- Mantener las manos, cables y objetos sueltos lejos del eje del motor.
- Utilizar gafas de seguridad durante pruebas de banco.
- Desconectar la batería antes de modificar conexiones.
- Verificar polaridad, conectores y aislamiento antes de energizar.
- Usar baterías LiPo en buen estado y cargadas con cargadores adecuados.
- No dejar baterías LiPo cargando sin supervisión.

SparkFun advierte que las baterías LiPo pueden incendiarse o explotar si son mal manipuladas, dañadas o expuestas a condiciones inadecuadas [5]. Adafruit también recomienda no sobrecargar, no sobredescargar y no exceder la corriente admisible de las baterías LiPo [6].

Este proyecto debe probarse inicialmente sin cargas peligrosas acopladas al eje del motor. El objetivo del capítulo es estudiar control seguro de ESC y rampas de arranque, no construir mecanismos de impacto o corte.

6.8. Fundamento teórico

6.8.1. Motor brushless

Un motor brushless es un motor eléctrico sin escobillas. Su rotor contiene imanes permanentes y el campo magnético del estator se controla electrónicamente. Esta conmutación requiere un ESC, que energiza las fases del motor de forma secuencial.

Entre sus ventajas se encuentran:

- Alta eficiencia.
- Alta velocidad.
- Menor desgaste mecánico que motores con escobillas.
- Buena relación potencia-peso.
- Tamaño compacto.

6.8.2. Diseño e implementación

El ESC recibe una señal de control y regula la potencia entregada al motor. En ESCs PWM convencionales, la señal se asemeja a la utilizada por servos RC. En muchos sistemas, el rango nominal se encuentra entre 1000 μ s y 2000 μ s, con una frecuencia de actualización típica cercana a la usada en servos. PX4 documenta el uso de salidas PWM para ESCs y servos, y en su procedimiento de calibración indica valores fijos de 1000 μ s y 2000 μ s para mínimo y máximo en ESCs compatibles [3, 4].

6.8.3. Rampa de aceleración

Una rampa de aceleración incrementa progresivamente la señal de control desde un valor inicial hasta una consigna deseada. Esto reduce cambios bruscos y permite observar el comportamiento del sistema. Si u_0 es la señal inicial, u_f la señal final y N el número de pasos, el incremento puede calcularse como:

$$\Delta_u = \frac{u_f - u_0}{N}$$

La señal en el paso k será:

$$u_k u_0 + \Delta_u$$

6.8.4. Estados de operación

El sistema utilizará una máquina de estados con cuatro estados [7]:

- **DESARMADO:** el ESC recibe señal mínima.
- **ARMADO:** el sistema está habilitado, pero el motor permanece en mínimo.
- **RAMPA:** la señal aumenta progresivamente hacia la consigna.
- **EMERGENCIA:** el sistema vuelve inmediatamente a señal mínima.

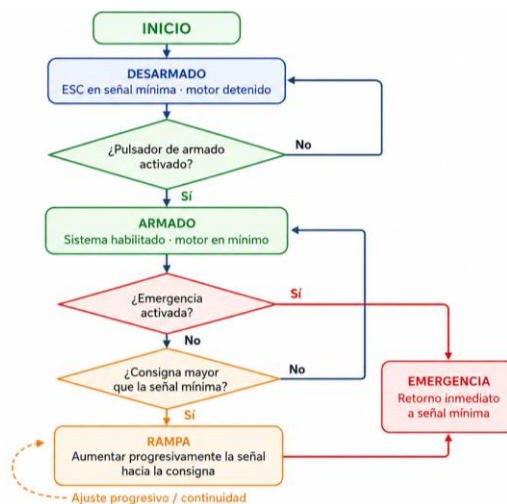


Figura 6.4: Máquina de estados del controlador seguro de motor brushless.

6.9. Diseño electrónico

Tabla 6.2: Asignación sugerida de pines para Arduino Nano.

Pin Arduino Nano	Conexión	Función
A0	Potenciómetro	Consigna de velocidad
D2	Pulsador de emergencia	Entrada de parada inmediata
D3	Señal del ESC	Salida de control tipo servo
D4	Pulsador de armado	Habilitación del sistema
D13	LED de estado	Indicación visual
5V	Circuito de control	Alimentación lógica
GND	GND común	Referencia común

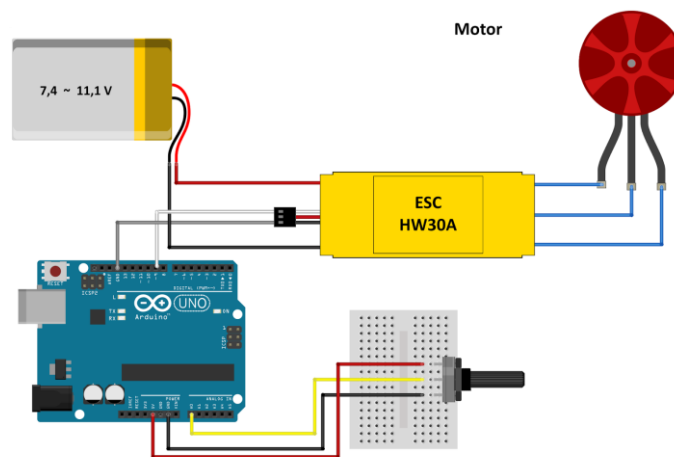


Figura 6.5: Diagrama de conexiones del controlador brushless con ESC.

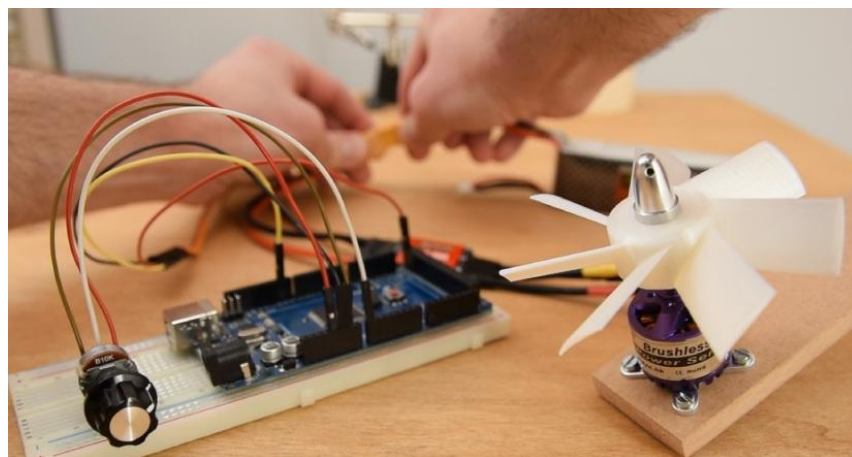


Figura 6.6: Montaje seguro para pruebas de motor brushless en banco.

6.10. Código del proyecto

El siguiente programa implementa un control seguro basado en armado, rampa de aceleración y emergencia. La señal mínima y máxima deben ajustarse de acuerdo con el ESC utilizado. Antes de conectar el motor, se recomienda verificar la señal mediante un servo de prueba o un analizador lógico si está disponible [8].

```
#include <Servo.h>

// PINES
const byte PIN_ESC = 3;
const byte PIN_POT = A0;
const byte PIN_ARMADO = 4;
const byte PIN_EMERGENCIA = 2;
const byte PIN_LED = 13;

// PARAMETROS DEL ESC
const int PULSO_MIN = 1000;
const int PULSO_MAX = 1800; // limite educativo inicial
const int PULSO_ARMADO = 1000;

// RAMPA
const int PASO_RAMPA = 5;
const unsigned long INTERVALO_RAMPA = 30;

// VARIABLES
Servo esc;

int pulsoActual = PULSO_MIN;
int pulsoObjetivo = PULSO_MIN;

unsigned long tiempoRampa = 0;

enum EstadoESC
{
    DESARMADO,
    ARMADO,
    RAMPA,
    EMERGENCIA
};

EstadoESC estadoActual = DESARMADO;

// FUNCIONES DE APOYO
bool botonPresionado(byte pin)
{
    return digitalRead(pin) == LOW;
```



```

}

void enviarPulsoSeguro(int pulso)
{
    pulso = constrain(pulso, PULSO_MIN, PULSO_MAX);
    esc.writeMicroseconds(pulso);
}

int leerConsigna()
{
    int adc = analogRead(PIN_POT);
    int pulso = map(adc, 0, 1023, PULSO_MIN, PULSO_MAX);
    return pulso;
}

void cambiarEstado(EstadoESC nuevoEstado)
{
    estadoActual = nuevoEstado;

    Serial.print("Estado: ");

    if (estadoActual == DESARMADO) Serial.println("DESARMADO");
    if (estadoActual == ARMADO) Serial.println("ARMADO");
    if (estadoActual == RAMPA) Serial.println("RAMPA");
    if (estadoActual == EMERGENCIA) Serial.println("EMERGENCIA");
}

// ESTADOS
void ejecutarDesarmado()
{
    pulsoActual = PULSO_MIN;
    enviarPulsoSeguro(pulsoActual);
    digitalWrite(PIN_LED, LOW);

    if (botonPresionado(PIN_ARMADO))
    {
        delay(300); // antirrebote simple
        cambiarEstado(ARMADO);
    }
}

void ejecutarArmado()
{
    pulsoActual = PULSO_ARMADO;
    enviarPulsoSeguro(pulsoActual);
    digitalWrite(PIN_LED, HIGH);
}

```

```

pulsoObjetivo = leerConsigna();

if (pulsoObjetivo > PULSO_MIN + 30)
{
    cambiarEstado(RAMPA);
}
}

void ejecutarRampa()
{
    pulsoObjetivo = leerConsigna();

    if (millis() - tiempoRampa >= INTERVALO_RAMPA)
    {
        tiempoRampa = millis();

        if (pulsoActual < pulsoObjetivo)
        {
            pulsoActual += PASO_RAMPA;
        }
        else if (pulsoActual > pulsoObjetivo)
        {
            pulsoActual -= PASO_RAMPA;
        }

        enviarPulsoSeguro(pulsoActual);
    }

    if (pulsoObjetivo <= PULSO_MIN + 30)
    {
        cambiarEstado(ARMADO);
    }
}

void ejecutarEmergencia()
{
    pulsoActual = PULSO_MIN;
    enviarPulsoSeguro(pulsoActual);
    digitalWrite(PIN_LED, millis() / 150 % 2);

    if (!botonPresionado(PIN_EMERGENCIA) && botonPresionado(PIN_ARMADO))
    {
        delay(500);
        cambiarEstado(ARMADO);
    }
}

```

```

// TELEMETRIA
void enviarTelemetry()
{
    static unsigned long t = 0;

    if (millis() - t >= 250)
    {
        t = millis();

        Serial.print("Pulso actual: ");
        Serial.print(pulsoActual);
        Serial.print(" us, objetivo: ");
        Serial.println(pulsoObjetivo);
    }
}

// SETUP Y LOOP
void setup()
{
    pinMode(PIN_ARMADO, INPUT_PULLUP);
    pinMode(PIN_EMERGENCIA, INPUT_PULLUP);
    pinMode(PIN_LED, OUTPUT);

    Serial.begin(115200);

    esc.attach(PIN_ESC);
    enviarPulsoSeguro(PULSO_MIN);

    Serial.println("Control seguro de ESC iniciado");
    Serial.println("Mantener pot en minimo antes de armar");
    delay(3000); // tiempo para inicializacion del ESC
}

void loop()
{
    if (botonPresionado(PIN_EMERGENCIA))
    {
        cambiarEstado(EMERGENCIA);
    }

    switch (estadoActual)
    {
        case DESARMADO:
            ejecutarDesarmado();
            break;

        case ARMADO:

```

```

    ejecutarArmado();
    break;

    case RAMPA:
        ejecutarRampa();
        break;

    case EMERGENCIA:
        ejecutarEmergencia();
        break;
}

enviarTelemetria();
}

```

El valor PULSO_MAX se ha limitado a 1800 μ s como medida educativa inicial. Para pruebas reales, el rango debe ajustarse según el ESC y el motor, siempre con supervisión, fijación mecánica adecuada y sin elementos peligrosos montados.

6.11. Procedimiento de prueba segura

6.11.1. Prueba 1: verificación sin batería del ESC

Antes de conectar la batería de potencia, cargar el programa en Arduino y verificar que los pulsadores funcionen correctamente mediante el monitor serial.

6.11.2. Prueba 2: señal de control

Conectar únicamente la señal del ESC y la referencia de tierra. Verificar que el Arduino envíe inicialmente el pulso mínimo.

6.11.3. Prueba 3: montaje mecánico

Fijar el motor brushless a una base rígida. No montar hélices, discos ni elementos giratorios peligrosos durante esta etapa.

6.11.4. Prueba 4: armado

Conectar la batería del ESC, mantener el potenciómetro en mínimo y presionar el pulsador de armado. El sistema debe permanecer en pulso mínimo.

6.11.5. Prueba 5: rampa progresiva

Aumentar lentamente el potenciómetro y verificar que la señal suba de forma gradual. Observar temperatura del motor, ESC y batería.

6.11.6. Prueba 6: emergencia

Presionar el botón de emergencia y verificar que la señal vuelva al valor mínimo.

Tabla 6.3: Registro sugerido de pruebas del controlador brushless.

Prueba	Condición	Resultado esperado	Resultado obtenido
Inicialización	Potenciómetro en mínimo	Pulso mínimo	
Armado	Pulsador presionado	Estado ARMADO	
Rampa	Aumentar consigna	Incremento progresivo	
Emergencia	Pulsador emergencia	Pulso mínimo	
Temperatura	Operación moderada	Sin sobrecalentamiento	

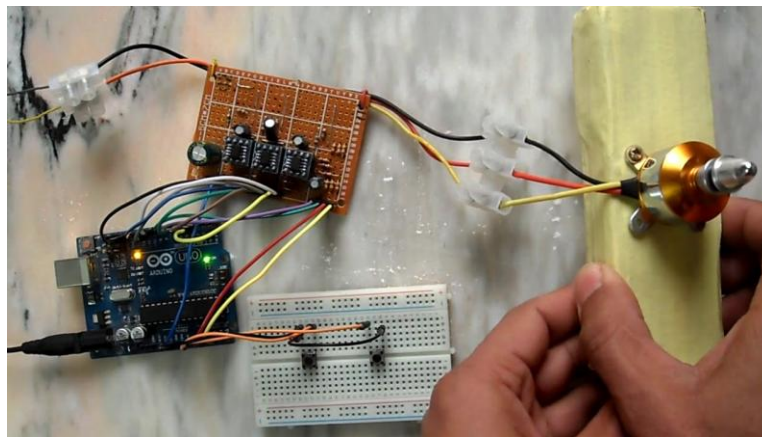


Figura 6.7: Pruebas experimentales del controlador brushless en banco.

6.12. Calibración del ESC

Algunos ESCs requieren calibrar el rango mínimo y máximo de señal. El procedimiento exacto depende del fabricante, por lo que debe consultarse la documentación del ESC específico. PX4 indica que, para ESCs PWM que soportan calibración, el proceso establece valores de entrada mínimos y máximos de referencia [4].

En un contexto educativo, se recomienda:

- No realizar calibración con elementos giratorios montados.
- Leer el manual del ESC antes de modificar parámetros.
- Usar valores conservadores al inicio.
- Confirmar el comportamiento de arranque con el motor fijado.

6.13. Análisis de resultados

El estudiante debe analizar:

- Tiempo requerido para alcanzar la consigna.

- Respuesta del motor ante diferentes pasos de rampa.
- Comportamiento térmico del ESC.
- Estabilidad de la señal de control.
- Efectividad del botón de emergencia.

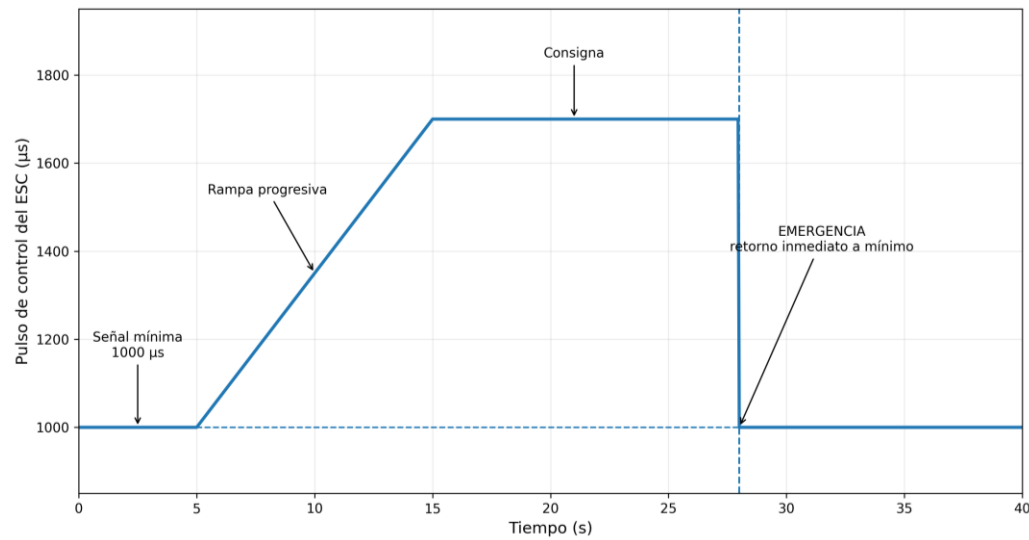


Figura 6.8: Perfil de rampa de control aplicado al ESC.

6.14. Errores frecuentes

Durante el desarrollo del proyecto pueden presentarse los siguientes errores:

- No compartir tierra entre Arduino y ESC.
- Conectar la batería de potencia antes de verificar la señal mínima.
- Usar `analogWrite()` en lugar de una señal tipo servo compatible con ESC.
- No fijar el motor a una base rígida.
- Usar un ESC incompatible con el motor.
- Aplicar cambios bruscos de consigna.
- No implementar parada de emergencia.
- Sobredescargar la batería LiPo durante pruebas.

6.15. Extensión del proyecto

Como extensión, se propone agregar medición de voltaje y corriente, reutilizando ideas del supervisor inteligente de seguridad desarrollado en el Capítulo 4. De esta forma, el sistema podría reducir automáticamente la consigna si detecta sobrecorriente o batería baja.

Otra extensión consiste en reemplazar el potenciómetro por comandos Bluetooth o una interfaz web en ESP32, permitiendo ajustar la rampa y el límite máximo de velocidad desde una aplicación.

- [3] PX4, “PWM Servos and ESCs”, PX4 User Guide. [En línea]. Disponible en: https://docs.px4.io/main/en/peripherals/pwm_escs_and_servo. [Accedido: 24-jun2026].
- [4] PX4, “ESC Calibration”, PX4 User Guide. [En línea]. Disponible en: https://docs.px4.io/main/en/advanced_config/esc_calibration. [Accedido: 24-jun-2026].
- [5] SparkFun, “Single Cell LiPo Battery Care”, SparkFun Learn. [En línea]. Disponible en: <https://learn.sparkfun.com/tutorials/single-cell-lipo-battery-care/battery-storage-and-handling>. [Accedido: 24-jun-2026].
- [6] Adafruit, “Li-Ion & LiPoly Batteries”, Adafruit Learning System. [En línea]. Disponible en: <https://learn.adafruit.com/li-ion-and-lipoly-batteries>. [Accedido: 24-jun-2026].
- [7] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2017.
- [8] M. Barr and A. Massa, *Programming Embedded Systems: With C and GNU Development Tools*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2006.

Capítulo 7. Sensores de Borde Inteligentes con Filtrado Temporal

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender el funcionamiento de sensores reflectivos utilizados para detección de línea y borde.
- Identificar causas frecuentes de lecturas falsas en sensores de borde de robots competitivos.
- Calibrar sensores reflectivos analógicos mediante lectura serial.
- Implementar filtrado digital simple para reducir ruido en señales analógicas.
- Aplicar histéresis para evitar cambios rápidos entre estados de detección.
- Implementar confirmación temporal para validar la detección real del borde.
- Integrar un módulo de sensores de borde inteligente en robots Minisumo, seguidores de línea o plataformas móviles.

7.1. Introducción

La detección confiable del borde es una función crítica en robots competitivos. En categorías como Minisumo, el robot debe evitar salir del dohyo mientras busca y ataca al adversario. En seguidores de línea, la lectura estable de sensores reflectivos permite mantener la trayectoria a altas velocidades. En plataformas móviles educativas, los sensores de borde pueden utilizarse para evitar caídas, delimitar zonas de movimiento o detectar cambios de superficie [5, 6].

Aunque el principio de funcionamiento de un sensor reflectivo parece sencillo, su comportamiento real puede verse afectado por múltiples factores: altura respecto al suelo, inclinación del sensor, iluminación ambiental, vibraciones, material de la pista, suciedad, ruido eléctrico, rebotes mecánicos y cambios rápidos de velocidad. Como resultado, el programa puede interpretar como borde una lectura falsa o, por el contrario, no reaccionar a tiempo ante una detección real. En robots Minisumo, una lectura falsa puede provocar que el robot retroceda innecesariamente y pierda una oportunidad de ataque. Una lectura tardía puede ser aún más grave, ya que el robot podría salir del dohyo y perder la ronda. Por esta razón, el diseño de la lectura de sensores no debe limitarse a comparar un valor analógico con un único umbral. Es recomendable incorporar técnicas de filtrado, histéresis y confirmación temporal [7].

Este capítulo desarrolla un sistema de *sensores de borde inteligentes* basado en Arduino Nano y sensores reflectivos analógicos. El proyecto permite calibrar sensores, filtrar lecturas, confirmar detecciones mediante una ventana de tiempo y generar señales confiables para el algoritmo principal del robot.

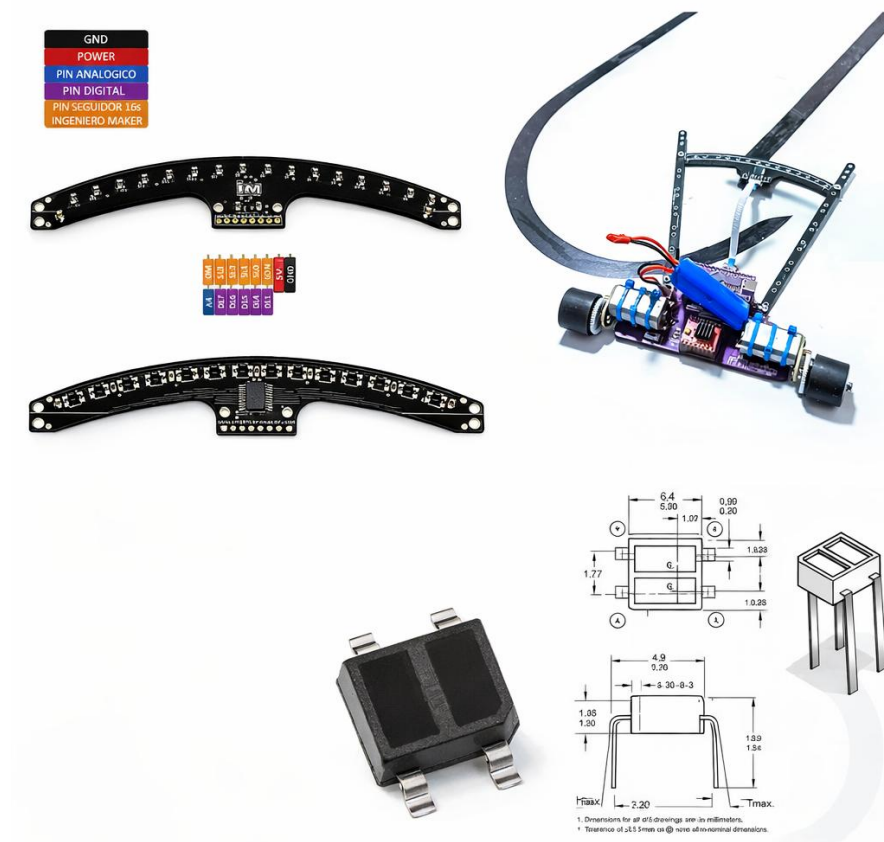


Figura 7.1: Sensores reflectivos instalados en un robot competitivo.

7.2. Planteamiento del problema

En muchos programas iniciales, la detección de borde se realiza mediante una condición directa:

```
if (analogRead(sensorBorde) > umbral)
{
    evadirBorde();
}
```

Esta estrategia puede funcionar en condiciones ideales, pero resulta vulnerable a falsas detecciones. Un solo pico de lectura, causado por vibración o ruido eléctrico, puede activar una maniobra de evasión. En un robot competitivo de alta velocidad, una evasión innecesaria puede desorganizar la estrategia, reducir el tiempo de ataque o entregar ventaja al oponente.

Los problemas más comunes son:

- Falsas detecciones por vibración del chasis.
- Cambios de lectura por variación de altura del sensor.
- Interferencia por iluminación ambiental.
- Ruido eléctrico producido por motores o drivers.

- Umbrales mal calibrados.
- Sensores con respuesta diferente entre sí.
- Lecturas inestables cerca del límite entre negro y blanco.

Un sensor de borde no debe evaluarse únicamente por una lectura instantánea. Para mejorar la confiabilidad, conviene analizar su comportamiento durante un pequeño intervalo de tiempo.

7.3. Descripción general del proyecto

El sistema propuesto utiliza cuatro sensores reflectivos analógicos conectados a un Arduino Nano. Dos sensores se ubican en la parte frontal del robot y dos pueden colocarse en los laterales o en la parte posterior, dependiendo de la categoría.

El programa realiza las siguientes tareas:

- Lectura analógica de sensores.
- Filtrado exponencial de cada lectura.
- Comparación mediante umbrales con histéresis.
- Confirmación temporal de detección.
- Generación de estado por sensor: libre, candidato o borde confirmado.
- Activación de salidas de alerta o señales para el controlador principal.
- Envío de telemetría por puerto serial.

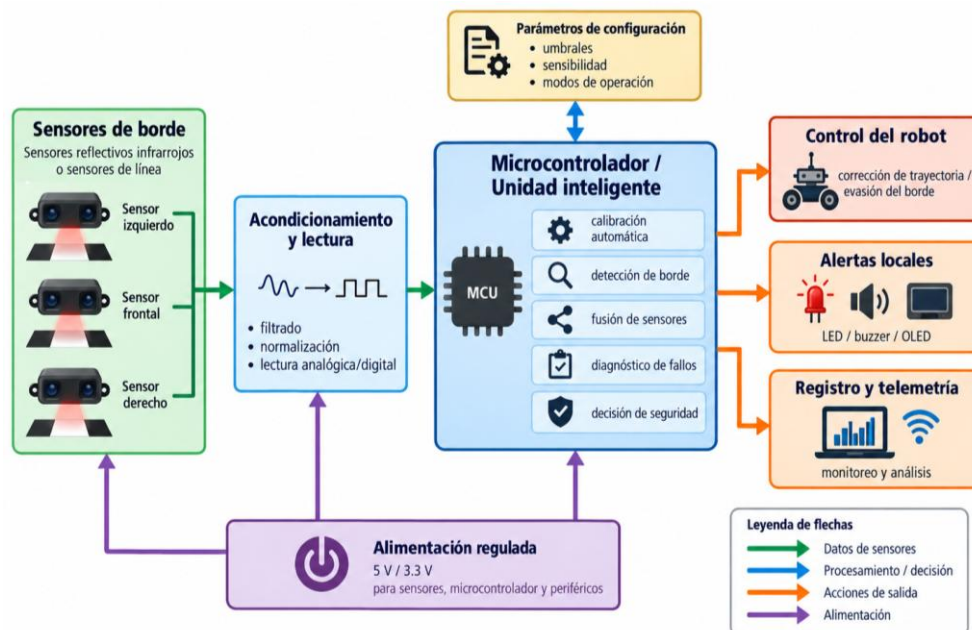


Figura 7.2: Arquitectura general del sistema de sensores de borde inteligentes.

7.4. Aplicación STEM

Este proyecto integra los componentes de la metodología STEM de la siguiente manera:

- **Ciencia:** análisis de reflexión de luz, influencia de la distancia, materiales y luz ambiental.
- **Tecnología:** uso de sensores reflectivos, microcontroladores y conversión analógico-digital.
- **Ingeniería:** diseño de una solución robusta para evitar salidas del dohyo o pérdida de trayectoria.
- **Matemáticas:** aplicación de umbrales, histéresis, filtros digitales y ventanas temporales.

7.5. Objetivos del proyecto

7.5.1. Objetivo general

Diseñar e implementar un sistema de detección de borde robusto para robots competitivos mediante sensores reflectivos, filtrado digital, histéresis y confirmación temporal.

7.5.2. Objetivos específicos

1. Calibrar sensores reflectivos analógicos sobre superficies claras y oscuras.
2. Implementar filtrado digital para suavizar lecturas.
3. Aplicar histéresis para evitar cambios rápidos entre detección y no detección.
4. Confirmar detecciones mediante una ventana temporal.
5. Generar salidas lógicas confiables para el algoritmo principal del robot.
6. Validar el sistema mediante pruebas sobre dohyo o pista de línea.

7.6. Materiales y herramientas

Tabla 7.1: Materiales principales para el sistema de sensores de borde inteligentes.

Elemento	Cantidad	Función
Arduino Nano	1	Procesamiento de señales de sensores
Sensores TCRT5000, QTR o similares	2 a 4	Detección de borde o línea
Resistencias y cables	Varios	Conexión eléctrica
LEDs indicadores	2 a 4	Visualización de detección
Buzzer opcional	1	Alarma durante pruebas
Robot Minisumo o seguidor de línea	1	Plataforma de validación
Dohyo o pista de prueba	1	Superficie de calibración
Computadora con Arduino IDE	1	Programación y monitoreo serial



Figura 7.3: Materiales utilizados para el sistema de sensores de borde inteligentes.

7.7. Fundamento teórico

7.7.1. Sensores reflectivos

Un sensor reflectivo integra un emisor de luz infrarroja y un receptor, generalmente un fototransistor. El emisor proyecta luz hacia la superficie y el receptor mide la cantidad de luz reflejada. Vishay describe el TCRT5000 como un sensor óptico reflectivo con emisor infrarrojo y fototransistor en un encapsulado que ayuda a bloquear luz visible [4].

La cantidad de luz reflejada depende de:

- Color de la superficie.
- Textura del material.
- Distancia entre sensor y superficie.
- Ángulo de montaje.
- Intensidad de iluminación externa.
- Corriente del emisor infrarrojo.

Los sensores QTR de Pololu son un ejemplo común de sensores de reflectancia utilizados en robots seguidores de línea y sumo. La documentación de Pololu diferencia sensores de salida analógica y sensores tipo RC, así como recomendaciones de uso y señales típicas [3, 5].

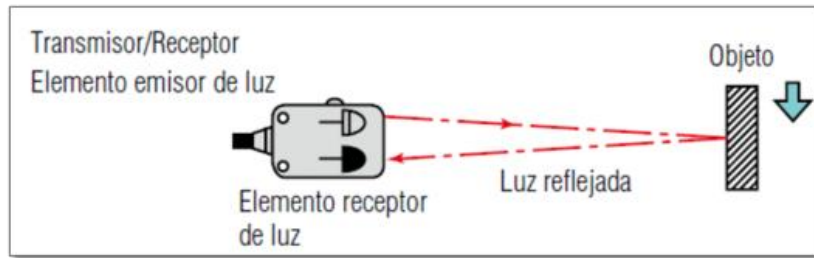


Figura 7.4: Principio de funcionamiento de un sensor reflectivo.

7.7.2. Conversión analógico-digital

Arduino permite leer señales analógicas mediante la función `analogRead()`. En placas clásicas como Arduino UNO o Nano basadas en AVR, la conversión analógico-digital es de 10 bits, lo que produce valores entre 0 y 1023 para un rango de tensión entre 0 V y el voltaje de referencia [1, 5].

Si se utiliza una referencia de 5 V, la resolución aproximada es:

$$\Delta V = \frac{5 \text{ V}}{1024} = 4.88 \text{ mV}$$

Por tanto, una lectura analógica puede convertirse a voltaje mediante:

$$V_{\text{sensor}} = \frac{\text{lectura}}{1023} \cdot V_{\text{ref}}$$

7.7.3. Umbral de detección

El umbral separa dos condiciones: superficie normal y borde. Si el sensor entrega valores mayores sobre superficie blanca, se puede definir:

$$\text{Borde} = \begin{cases} 1, & \text{si } \text{lectura} > \text{umbral} \\ 0, & \text{si } \text{lectura} \leq \text{umbral} \end{cases}$$

Sin embargo, cuando la lectura fluctúa cerca del umbral, el estado puede cambiar rápidamente entre 0 y 1. Para evitarlo, se introduce histéresis.

7.7.4. Histéresis

La histéresis utiliza dos umbrales:

- Umbral alto: confirma posible borde.
- Umbral bajo: confirma retorno a superficie normal.

De esta forma, el sistema no cambia de estado inmediatamente cuando la lectura oscila cerca de un único valor.

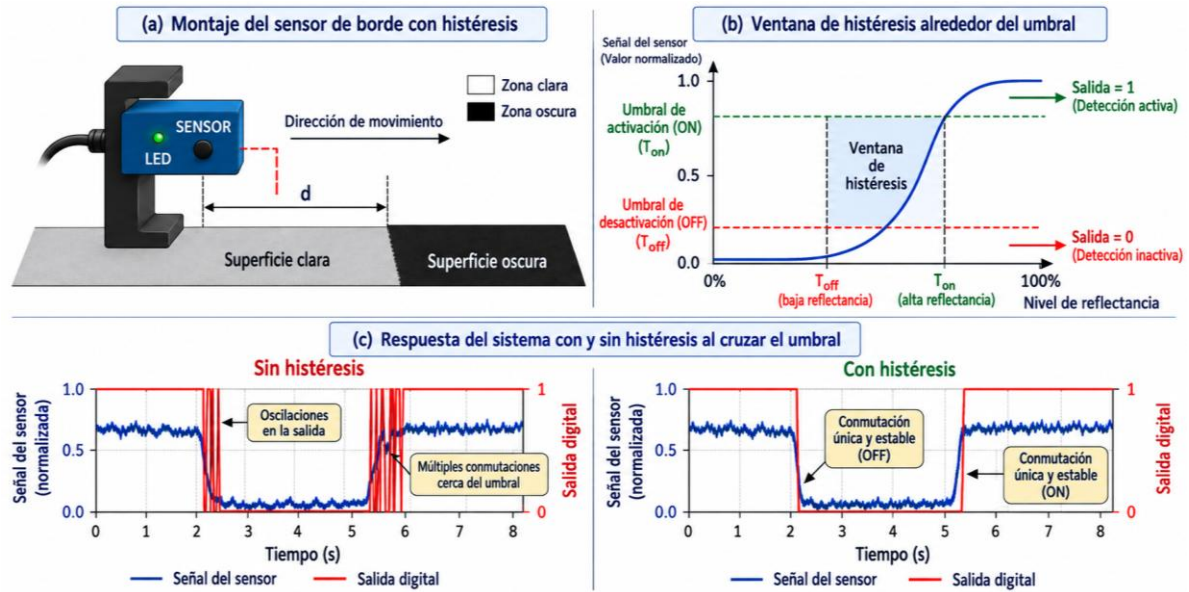


Figura 7.5: Uso de histéresis para evitar oscilaciones cerca del umbral.

7.7.5. Confirmación temporal

La confirmación temporal exige que la condición de borde se mantenga durante un intervalo mínimo antes de activar la detección final. Esta técnica evita reaccionar ante picos aislados.

Si el sensor supera el umbral alto en el instante t_0 , el borde se confirma solo si:

$$t - t_0 \geq T_{confirmacion}$$

donde $T_{confirmacion}$ es la ventana temporal de validación.

Arduino proporciona la función `millis()`, que devuelve el número de milisegundos transcurridos desde que la placa comenzó a ejecutar el programa [6]. Esta función permite implementar temporización sin bloquear el ciclo principal [2].

7.7.6. Filtro exponencial

El filtro exponencial suaviza la lectura del sensor combinando la medición actual con el valor filtrado anterior:

$$y_k = \alpha x_k + (1 - \alpha)y_{k-1}$$

donde:

- x_k es la nueva medición.
- y_k es el valor filtrado.
- α es el factor de suavizado.

Un valor pequeño de α produce una señal más suave pero más lenta. Un valor grande responde más rápido, pero filtra menos ruido.

7.8. Diseño electrónico

En este proyecto se utilizarán cuatro sensores reflectivos analógicos. Si el lector solo dispone de dos sensores, puede usar únicamente los sensores frontales.

Tabla 7.2: Asignación sugerida de pines para sensores de borde.

Pin Arduino Nano	Conexión	Función
A0	Sensor frontal izquierdo	Detección de borde frontal izquierdo
A1	Sensor frontal derecho	Detección de borde frontal derecho
A2	Sensor lateral o posterior izquierdo	Detección auxiliar izquierda
A3	Sensor lateral o posterior derecho	Detección auxiliar derecha
D4	LED borde izquierdo	Indicador visual
D5	LED borde derecho	Indicador visual
D8	Salida de borde global	Señal hacia controlador principal
D13	LED integrado	Depuración

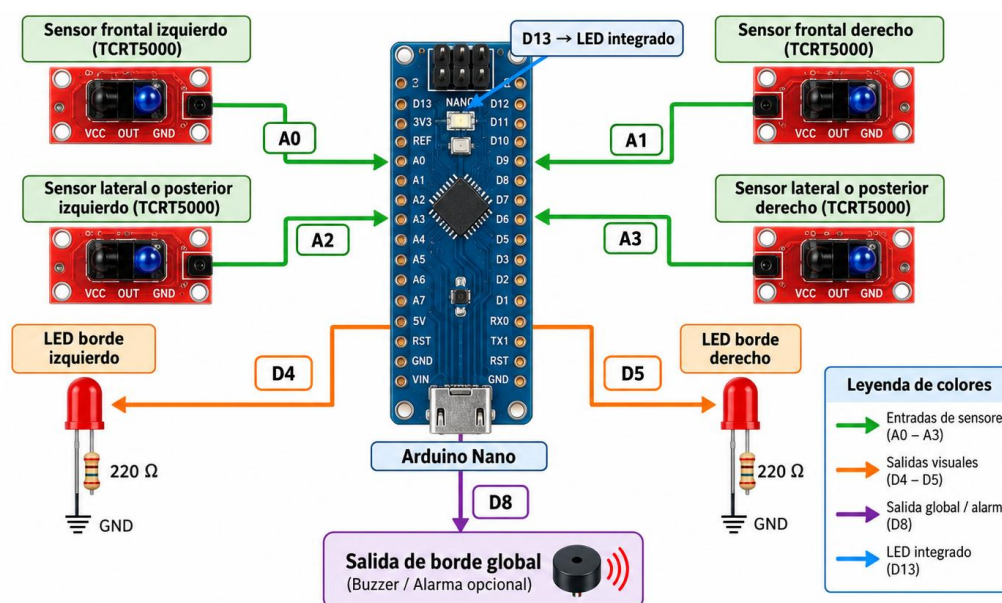


Figura 7.6: Diagrama de conexiones del sistema de sensores de borde inteligentes.

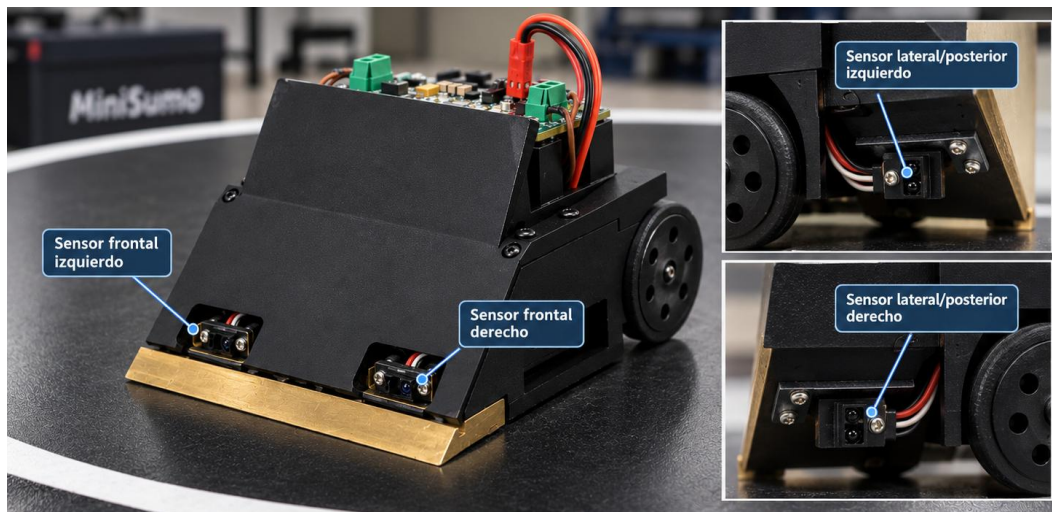


Figura 7.7: Montaje físico de sensores de borde en un robot competitivo.

7.9. Calibración inicial

Antes de aplicar filtrado o histéresis, se debe conocer el rango de lectura de cada sensor. Para ello se utiliza un programa de lectura serial.

```
void setup()
{
  Serial.begin(115200);
}

void loop()
{
  Serial.print("S0:");
  Serial.print(analogRead(A0));

  Serial.print(" S1:");
  Serial.print(analogRead(A1));

  Serial.print(" S2:");
  Serial.print(analogRead(A2));

  Serial.print(" S3:");
  Serial.println(analogRead(A3));

  delay(100);
}
```

Se recomienda registrar valores en varias condiciones:

- Sensor sobre superficie negra.
- Sensor sobre línea blanca.

- Sensor a diferente altura.
- Robot quieto.
- Robot vibrando con motores encendidos.
- Pista con iluminación normal y con iluminación variable.

Tabla 7.3: Formato sugerido para calibración de sensores de borde.

Sensor	Negro	Blanco	Umbral bajo	Umbral alto
Frontal izquierdo				
Frontal derecho				
Auxiliar izquierdo				
Auxiliar derecho				

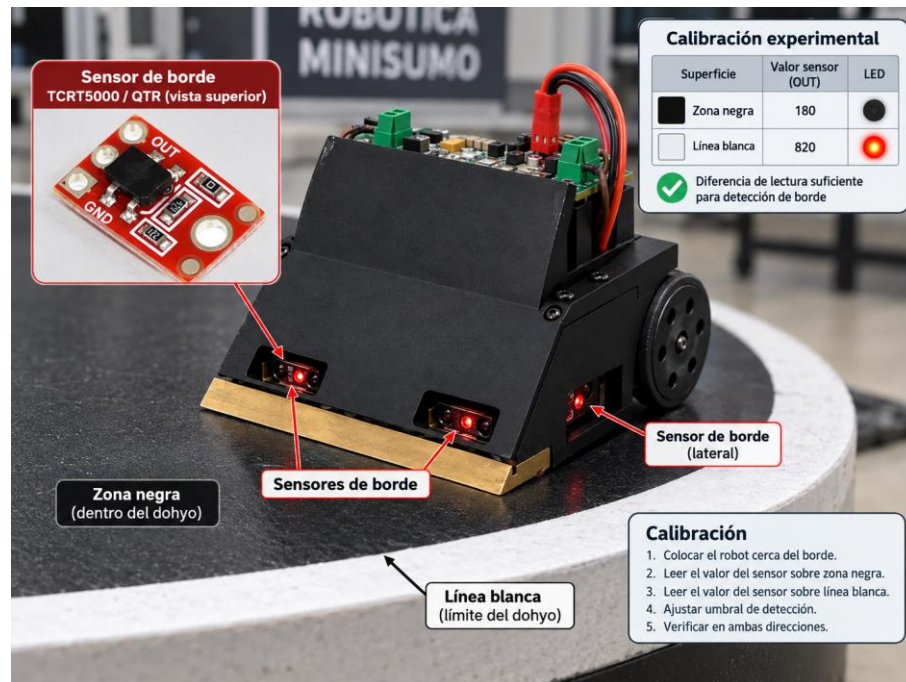


Figura 7.8: Calibración experimental de sensores de borde sobre dohyo.

7.10. Organización modular del programa

El programa principal se organiza en los siguientes bloques:

- Definición de pines y parámetros.
- Lectura analógica de sensores.
- Filtrado exponencial.
- Evaluación con histéresis.
- Confirmación temporal.
- Activación de salidas y telemetría.

Esta organización permite integrar el módulo en un Minisumo o seguidor de línea sin mezclar la lectura de sensores con la estrategia completa del robot.

7.11. Código del proyecto

El siguiente programa implementa lectura, filtrado, histéresis y confirmación temporal para cuatro sensores reflectivos.

```
/*
  Sensores de borde inteligentes
  Plataforma: Arduino Nano
  Sensores: reflectivos analogicos
*/

// =====
// CONFIGURACION GENERAL
// =====
const byte NUM_SENSORES = 4;

const byte pinesSensores[NUM_SENSORES] = {A0, A1, A2, A3};

// Indicadores
const byte LED_IZQ = 4;
const byte LED_DER = 5;
const byte SALIDA_BORDE_GLOBAL = 8;
const byte LED_ESTADO = 13;

// =====
// PARAMETROS AJUSTABLES
// =====
// Ajustar segun calibracion real
const int UMBRAL_ALTO = 650;
const int UMBRAL_BAJO = 520;

// Filtro exponencial
const float ALPHA = 0.35;

// Tiempo minimo para confirmar borde
const unsigned long TIEMPO_CONFIRMACION = 25;

// =====
// ESTRUCTURA DE SENSOR
// =====
enum EstadoSensor
{
  LIBRE,
  CANDIDATO,
```

```

BORDE_CONFIRMADO
};

struct SensorBorde
{
    int crudo;
    float filtrado;
    EstadoSensor estado;
    unsigned long tiempoCandidato;
};

SensorBorde sensores[NUM_SENSORES];

// =====
// INICIALIZACION
// =====
void inicializarSensores()
{
    for (byte i = 0; i < NUM_SENSORES; i++)
    {
        sensores[i].crudo = analogRead(pinesSensores[i]);
        sensores[i].filtrado = sensores[i].crudo;
        sensores[i].estado = LIBRE;
        sensores[i].tiempoCandidato = 0;
    }
}

// =====
// LECTURA Y FILTRADO
// =====
void leerYFiltrarSensores()
{
    for (byte i = 0; i < NUM_SENSORES; i++)
    {
        sensores[i].crudo = analogRead(pinesSensores[i]);

        sensores[i].filtrado =
        ALPHA * sensores[i].crudo + (1.0 - ALPHA) * sensores[i].filtrado;
    }
}

// =====
// EVALUACION CON HISTERESIS Y TIEMPO
// =====
void evaluarSensores()
{
    for (byte i = 0; i < NUM_SENSORES; i++)

```

```

{
    float valor = sensores[i].filtrado;

    switch (sensores[i].estado)
    {
        case LIBRE:
            if (valor > UMBRAL_ALTO)
            {
                sensores[i].estado = CANDIDATO;
                sensores[i].tiempoCandidato = millis();
            }
            break;

        case CANDIDATO:
            if (valor < UMBRAL_BAJO)
            {
                sensores[i].estado = LIBRE;
            }
            else if (millis() - sensores[i].tiempoCandidato >= TIEMPO_CONFIRMACION)
            {
                sensores[i].estado = BORDE_CONFIRMADO;
            }
            break;

        case BORDE_CONFIRMADO:
            if (valor < UMBRAL_BAJO)
            {
                sensores[i].estado = LIBRE;
            }
            break;
    }
}

// =====
// CONSULTA DE ESTADOS
// =====
bool bordeIzquierdo()
{
    return sensores[0].estado == BORDE_CONFIRMADO ||
        sensores[2].estado == BORDE_CONFIRMADO;
}

bool bordeDerecho()
{
    return sensores[1].estado == BORDE_CONFIRMADO ||
        sensores[3].estado == BORDE_CONFIRMADO;
}

```

```

}

bool bordeGlobal()
{
    return bordeIzquierdo() || bordeDerecho();
}

// =====
// SALIDAS Y TELEMETRIA
// =====
void actualizarSalidas()
{
    digitalWrite(LED_IZQ, bordeIzquierdo());
    digitalWrite(LED_DER, bordeDerecho());
    digitalWrite(SALIDA_BORDE_GLOBAL, bordeGlobal());
    digitalWrite(LED_ESTADO, bordeGlobal());
}

void enviarTelemetria()
{
    static unsigned long tiempoSerial = 0;

    if (millis() - tiempoSerial >= 100)
    {
        tiempoSerial = millis();

        for (byte i = 0; i < NUM_SENSORES; i++)
        {
            Serial.print("S");
            Serial.print(i);
            Serial.print(":");
            Serial.print(sensores[i].crudo);
            Serial.print("/");
            Serial.print(sensores[i].filtrado, 1);
            Serial.print(" E:");
            Serial.print(sensores[i].estado);
            Serial.print(" ");
        }

        Serial.print(" BORDE_GLOBAL:");
        Serial.println(bordeGlobal());
    }
}

// =====
// SETUP Y LOOP
// =====

```

```

void setup()
{
  pinMode(LED_IZQ, OUTPUT);
  pinMode(LED_DER, OUTPUT);
  pinMode(SALIDA_BORDE_GLOBAL, OUTPUT);
  pinMode(LED_ESTADO, OUTPUT);

  Serial.begin(115200);

  inicializarSensores();

  Serial.println("Sistema de sensores de borde inteligentes iniciado");
}

void loop()
{
  leerYFiltrarSensores();
  evaluarSensores();
  actualizarSalidas();
  enviarTelemetria();
}

```

Los valores UMBRAL_ALTO, UMBRAL_BAJO, ALPHA y TIEMPO_CONFIRMACION deben ajustarse experimentalmente. No existe un valor universal válido para todos los sensores, pistas y robots.

7.12. Integración con un robot Minisumo

En un Minisumo, la salida global del módulo puede conectarse al controlador principal o utilizarse directamente dentro del mismo Arduino. Cuando se detecta borde izquierdo, el robot debe retroceder y girar hacia la derecha. Cuando se detecta borde derecho, debe retroceder y girar hacia la izquierda.

```

if (bordeIzquierdo())
{
  mover(-180, -180);
  delay(120);
  mover(180, -180);
}
else if (bordeDerecho())
{
  mover(-180, -180);
  delay(120);
  mover(-180, 180);
}

```

```

else
{
    ejecutarEstrategiaNormal();
}

```

Aunque el ejemplo utiliza retardos cortos para simplificar la maniobra, en robots competitivos avanzados se recomienda implementar estas acciones mediante máquina de estados no bloqueante.

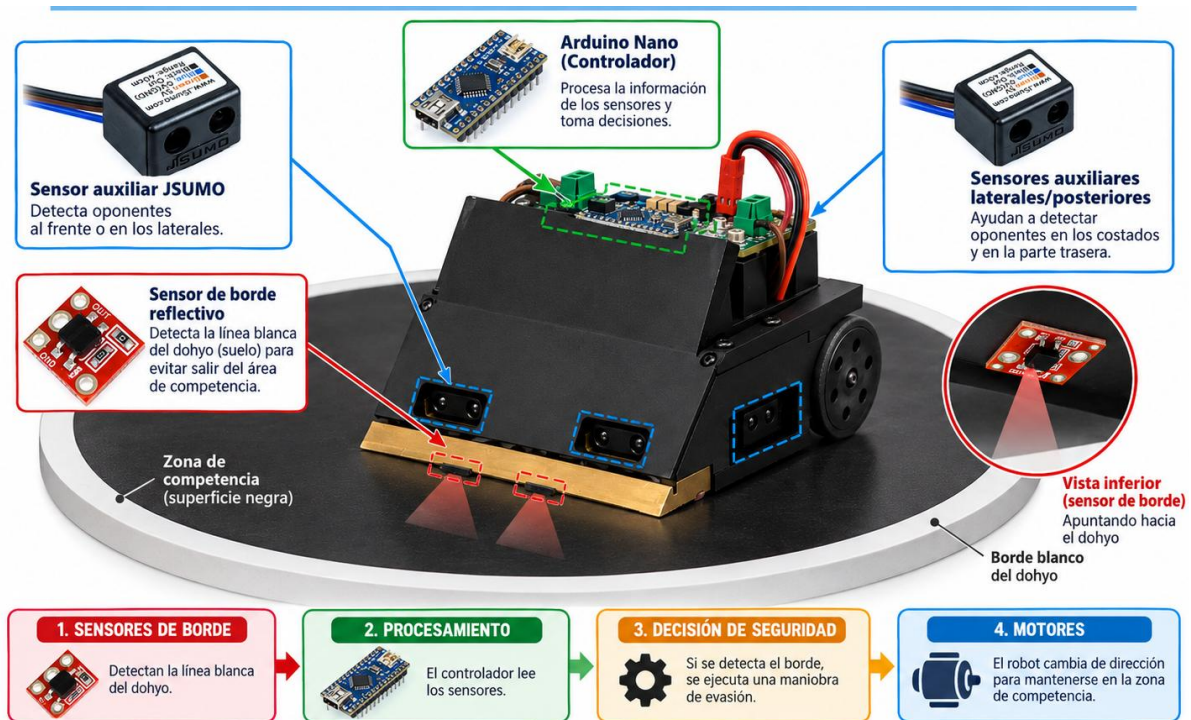


Figura 7.9: Integración del sistema de detección de borde en un robot Minisumo.

7.13. Integración con un seguidor de línea

En un seguidor de línea, el mismo principio puede utilizarse para descartar lecturas falsas de sensores laterales, detectar pérdida de pista o validar cambios bruscos en la trayectoria. En robots de alta velocidad, las vibraciones pueden hacer que una barra de sensores cambie momentáneamente su distancia respecto a la pista, generando valores erráticos.

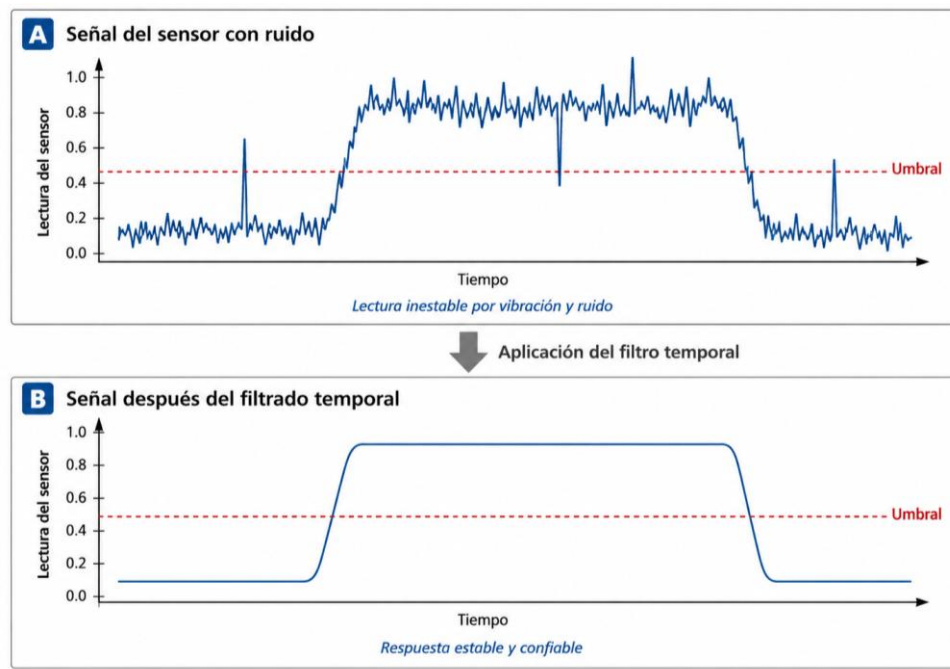


Figura 7.10: Aplicación del filtrado temporal en sensores de un robot seguidor de línea.

7.14. Pruebas experimentales

La validación del sistema debe realizarse en varias etapas.

7.14.1. Prueba 1: lectura en reposo

Colocar los sensores sobre superficie negra y observar estabilidad de valores en el monitor serial.

7.14.2. Prueba 2: detección de borde blanco

Mover lentamente el robot hacia la línea blanca y verificar que el estado pase de LIBRE a CANDIDATO y luego a BORDE_CONFIRMADO.

Prueba 3: ruido por vibración

Encender motores con el robot suspendido o sobre una base segura y observar si las vibraciones generan falsas detecciones.

Prueba 4: prueba dinámica sobre dohyo

Permitir que el robot avance hacia el borde y verificar si el sistema confirma la detección con suficiente anticipación.

Tabla 7.4: Registro sugerido de pruebas del sistema de borde inteligente.

Prueba	Condición	Resultado esperado	Resultado obtenido
Reposo negro	Robot quieto	Estado LIBRE	
Línea blanca	Sensor sobre borde	BORDE_CONFIRMADO	
Vibración	Motores encendidos	Sin falsas detecciones	
Dohyo dinámico	Avance hacia borde	Evasión oportuna	

7.15. Ajuste de parámetros

Los parámetros deben ajustarse según el comportamiento observado.

7.15.1. Umbral alto

Si el sistema no detecta el borde, el umbral alto puede estar demasiado elevado. Debe reducirse cuidadosamente hasta que la detección sea confiable.

7.15.2. Umbral bajo

Si el sistema permanece detectando borde después de volver a la superficie negra, el umbral bajo puede estar demasiado bajo o la histéresis puede ser excesiva.

7.15.3. Factor de filtro

Si el sistema responde muy lento, puede aumentarse α . Si hay mucho ruido, puede reducirse.

7.15.4. Tiempo de confirmación

Si hay falsas detecciones, puede aumentarse el tiempo de confirmación. Si el robot se sale del dohyo antes de reaccionar, debe reducirse el tiempo o mejorar la posición física del sensor.

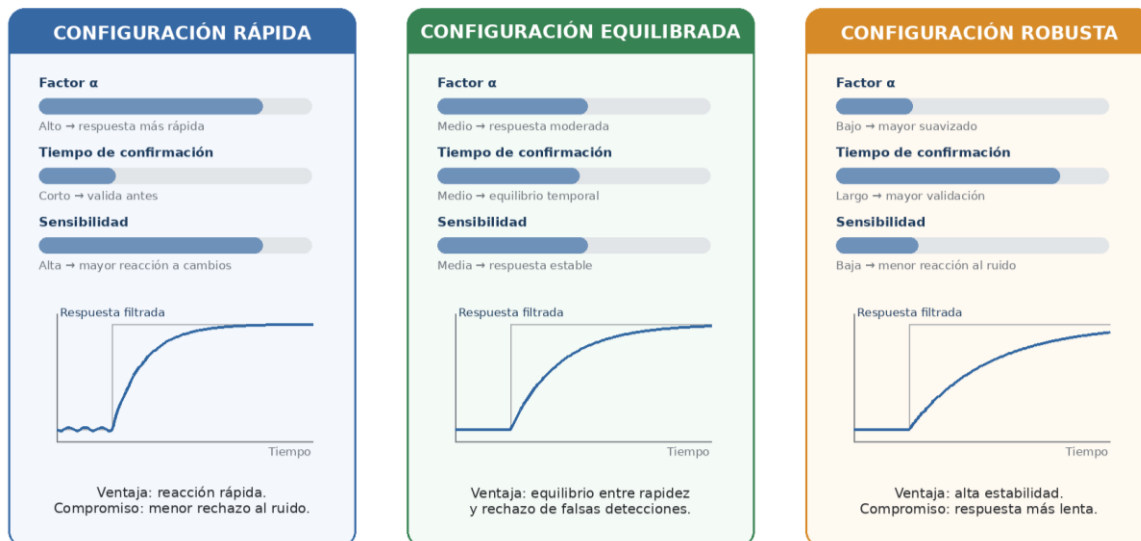


Figura 7.11: Comparación conceptual de configuraciones de filtrado temporal.

7.16. Errores frecuentes

Durante el desarrollo pueden presentarse los siguientes errores:

- Sensores instalados demasiado lejos del suelo.
- Sensores inclinados de forma diferente entre sí.
- Umbrales definidos sin realizar calibración previa.
- Cables de sensores pasando junto a cables de motor.
- Ausencia de tierra común.
- Filtro demasiado lento para la velocidad del robot.
- Tiempo de confirmación excesivo.
- Falta de pruebas con motores encendidos.

7.17. Proyecto de extensión

Como extensión, se propone crear un sistema de calibración automática. Durante los primeros segundos, el robot podría registrar valores mínimos y máximos de cada sensor sobre superficies conocidas y calcular automáticamente los umbrales de trabajo.

Otra mejora consiste en enviar los valores por Bluetooth a una aplicación móvil, permitiendo visualizar lecturas en tiempo real y ajustar umbrales sin reprogramar el microcontrolador.

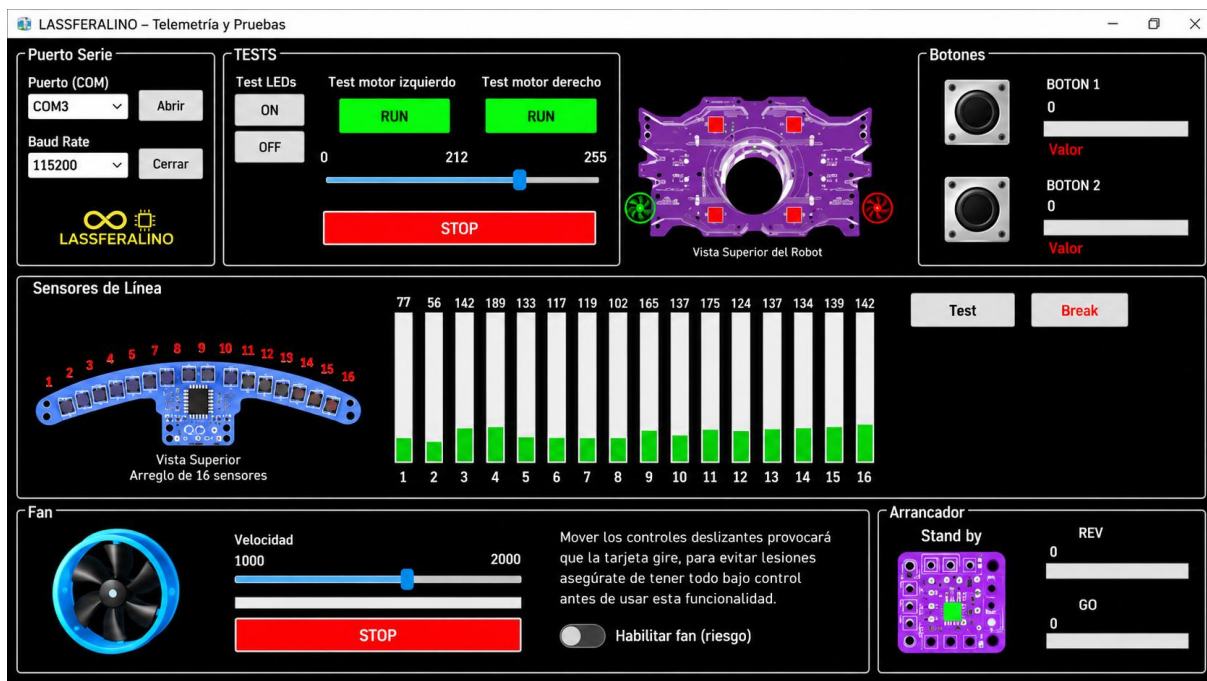


Figura 7.12: Extensión del sistema mediante calibración y monitoreo inalámbrico.

Actividades de aprendizaje

1. Calibre dos sensores reflectivos sobre superficie negra y línea blanca.
2. Determine valores adecuados de umbral alto y bajo para su pista.
3. Modifique el valor de ALPHA y observe el efecto sobre la respuesta del sistema.
4. Compare el comportamiento del sistema con y sin confirmación temporal.
5. Integre el módulo de borde en un Minisumo o seguidor de línea.
6. Diseñe una versión con calibración automática de umbrales.

Preguntas de reflexión

1. ¿Por qué un único umbral puede generar falsas detecciones?
2. ¿Qué ventaja ofrece la histéresis en sensores de borde?
3. ¿Cómo influye la altura del sensor sobre la lectura?
4. ¿Qué ocurre si el tiempo de confirmación es demasiado grande?
5. ¿Por qué es importante probar sensores con los motores encendidos?
6. ¿Cómo aplicaría este sistema a una barra Phoenix de sensores en un seguidor de línea?

Referencias

- [1] Arduino, “analogRead()”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/language-reference/en/functions/analog-io/analogRead/>. [Accedido: 24-jun-2026].
- [2] Arduino, “millis()”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/language-reference/en/functions/time/millis/>. [Accedido: 24-jun-2026].
- [3] Pololu, “QTR Reflectance Sensor Application Note”, Pololu Documentation. [En línea]. Disponible en: <https://www.pololu.com/docs/0J13>. [Accedido: 24-jun-2026].
- [4] Vishay Semiconductors, “TCRT5000, TCRT5000L Reflective Optical Sensor with Transistor Output”, Datasheet. [En línea]. Disponible en: <https://www.vishay.com/docs/83760/tcrt5000.pdf>. [Accedido: 24-jun-2026].
- [5] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed. Cham, Switzerland: Springer, 2016.
- [6] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.
- [7] D. Cook, *Robot Building for Beginners*, 4th ed. Berkeley, CA, USA: Apress, 2020.

Capítulo 8. Cinturón Interactivo RFID con Efectos de Luz y Sonido

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender el uso de tecnología RFID para identificación de tarjetas, llaveros o accesorios interactivos.
- Integrar un lector RFID RC522 con Arduino mediante comunicación SPI.
- Controlar LEDs direccionables WS2812B para generar secuencias visuales.
- Reproducir efectos sonoros mediante un módulo DFPlayer Mini y una memoria microSD.
- Diseñar una máquina de estados para organizar modos de espera, lectura, activación y error.
- Construir un wearable educativo inspirado en props tecnológicos, cosplay y sistemas de interacción física.

8.1. Introducción

La electrónica interactiva permite transformar objetos cotidianos en sistemas capaces de responder al usuario mediante luz, sonido, movimiento o comunicación inalámbrica. Esta área combina creatividad, diseño, programación y hardware embebido, convirtiéndose en una herramienta atractiva para educación STEM, prototipado, cosplay, exhibiciones, museos interactivos y demostraciones tecnológicas.

En este capítulo se desarrolla un cinturón interactivo RFID con efectos de luz y sonido. El proyecto se inspira en dispositivos de transformación ficticios presentes en series de ciencia ficción, tokusatsu, videojuegos y cultura maker; sin embargo, su propósito es completamente educativo. El sistema permite que diferentes tarjetas RFID activen efectos visuales y sonoros distintos, simulando modos, perfiles o configuraciones especiales.

El proyecto utiliza un Arduino Nano como controlador principal, un lector RFID RC522 para identificar tarjetas, una tira de LEDs direccionables WS2812B para generar efectos lumínicos y un módulo DFPlayer Mini para reproducir sonidos desde una memoria microSD. El resultado es un wearable tecnológico que puede adaptarse a presentaciones estudiantiles, ferias de robótica, actividades de vinculación, clubes maker o exhibiciones del Club de Robótica UTE.

A diferencia de los capítulos anteriores, centrados principalmente en robótica competitiva, este proyecto introduce el bloque de wearables tecnológicos. Estos sistemas permiten aplicar los mismos principios de electrónica, programación y diseño modular en objetos portables e interactivos.



Figura 8.1: Prototipo de cinturón interactivo con efectos de luz y sonido.

8.2. Planteamiento del problema

En muchos proyectos educativos de electrónica, los estudiantes aprenden a encender LEDs, leer sensores o reproducir sonidos como ejercicios separados. Sin embargo, pocas veces integran esos conocimientos en un objeto interactivo con propósito narrativo, estético y funcional.

Un wearable interactivo plantea retos interesantes:

- El sistema debe reconocer diferentes entradas físicas.
- Debe responder con efectos visuales y sonoros coherentes.
- Debe organizarse mediante estados para evitar respuestas desordenadas.
- Debe consumir poca energía y ser portable.
- Debe tener una estructura mecánica cómoda y segura.
- Debe permitir personalización de modos, colores y sonidos.

Este proyecto resuelve el problema mediante una arquitectura simple: cada tarjeta RFID representa un modo de activación. Cuando el sistema detecta una tarjeta autorizada, ejecuta una secuencia de LEDs y reproduce un archivo de audio asociado.

Un proyecto wearable no debe verse solo como decoración. Es una oportunidad para integrar identificación digital, diseño de interacción, electrónica portátil, programación por estados y fabricación digital.

8.3. Descripción general del proyecto

El cinturón interactivo se compone de los siguientes bloques:

- **Identificación:** lector RFID RC522 y tarjetas o llaveros RFID.
- **Control:** Arduino Nano.

- **Salida visual:** tira de LEDs direccionables WS2812B.
- **Salida sonora:** módulo DFPlayer Mini con memoria microSD y parlante.
- **Alimentación:** batería portátil o banco de energía de 5 V.
- **Estructura:** carcasa impresa en 3D, cinturón o soporte rígido.

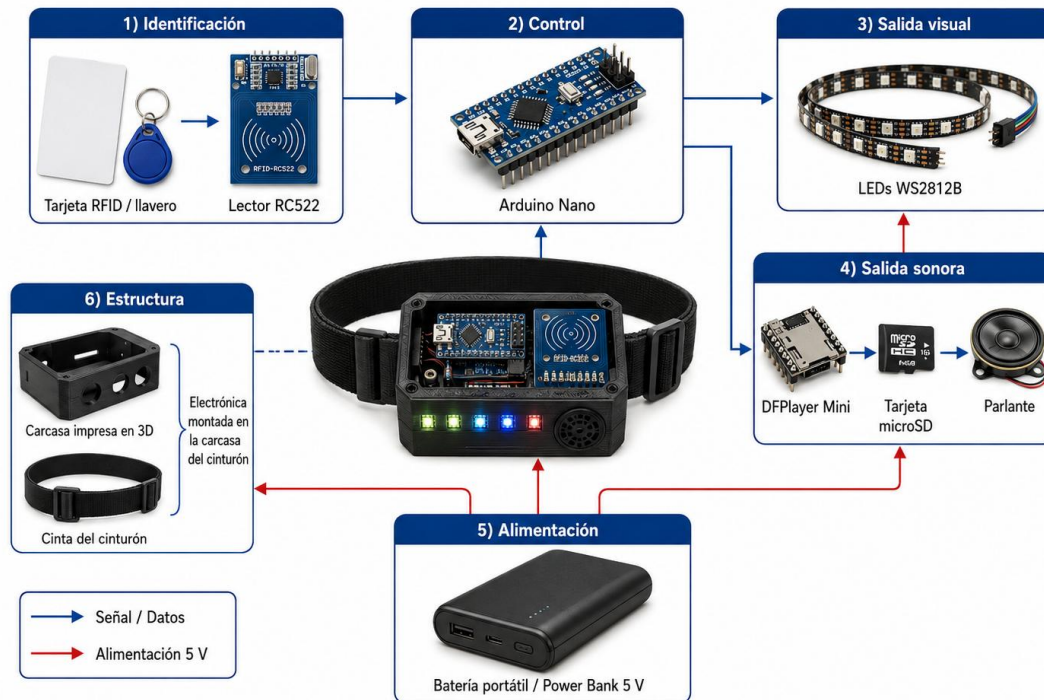


Figura 8.2: Arquitectura general del cinturón interactivo RFID.

8.4. Aplicación STEM

Este proyecto integra las áreas STEM de la siguiente manera:

- **Ciencia:** principios de comunicación por radiofrecuencia, luz, sonido y energía.
- **Tecnología:** uso de RFID, LEDs direccionables, audio digital y microcontroladores.
- **Ingeniería:** diseño de un objeto portable, distribución de energía y organización del software.
- **Matemáticas:** secuencias temporales, mapeo de identificadores, conteo de LEDs y temporización de efectos.

8.5. Objetivos del proyecto

8.5.1. Objetivo general

Diseñar e implementar un cinturón interactivo basado en RFID, capaz de activar efectos de luz y sonido mediante tarjetas o llaveros identificables por Arduino.

8.5.2. Objetivos específicos

- 1. Conectar un lector RC522 a Arduino Nano mediante comunicación SPI.
- 2. Leer el UID de tarjetas RFID.
- 3. Asociar tarjetas autorizadas con diferentes modos de activación.
- 4. Controlar una tira de LEDs WS2812B mediante secuencias programadas.
- 5. Reproducir archivos MP3 mediante un módulo DFPlayer Mini.
- 6. Organizar el comportamiento del wearable mediante una máquina de estados.
- 7. Proponer una carcasa o estructura portable para el sistema.

8.6. Materiales y herramientas

Tabla 8.1: Materiales principales para el cinturón interactivo RFID.

Elemento	Cantidad	Función
Arduino Nano	1	Control principal del sistema
Lector RFID RC522	1	Lectura de tarjetas RFID
Tarjetas o llaveros RFID	2 o más	Activación de modos
Tira WS2812B	1	Efectos de iluminación
DFPlayer Mini	1	Reproducción de audio MP3
Memoria microSD	1	Almacenamiento de sonidos
Parlante 3 W aprox.	1	Salida sonora
Batería USB o power bank	1	Alimentación portable
Resistencia 330 – 470 Ω	1	Protección de línea de datos LED
Capacitor 470 – 1000 μ F	1	Estabilización de alimentación LED
Carcasa o cinturón	1	Soporte mecánico

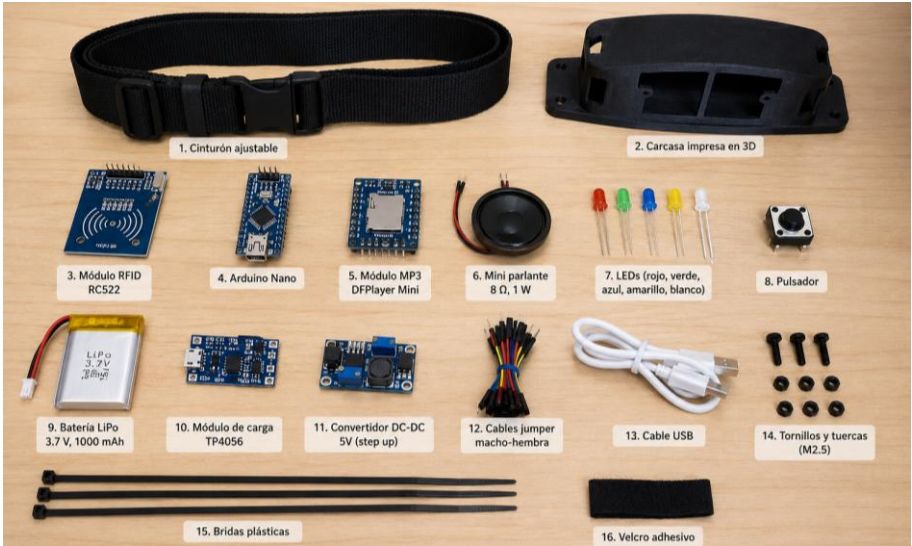


Figura 8.3: Materiales utilizados para construir el cinturón interactivo RFID.

8.7. Fundamento teórico

8.7.1. Tecnología RFID

RFID significa *Radio Frequency Identification*. Esta tecnología permite identificar objetos mediante etiquetas o tarjetas que almacenan un código. El lector genera un campo de radiofrecuencia y la tarjeta responde con información que puede ser leída por el sistema.

El módulo RC522 está basado en el circuito MFRC522 de NXP, orientado a aplicaciones de lectura y escritura sin contacto a 13.56 MHz [1]. En proyectos con Arduino, este módulo suele utilizarse con la librería MFRC522, que facilita la lectura del UID de tarjetas compatibles [2].

El UID es un identificador único o semipermanente de la tarjeta. En este proyecto se utilizará para seleccionar diferentes efectos.

8.7.2. Comunicación SPI

El lector RC522 se comunica con Arduino mediante el protocolo SPI. SPI utiliza líneas separadas para datos de entrada, datos de salida, reloj y selección de dispositivo [6].

Tabla 8.2: Conexiones típicas entre RC522 y Arduino Nano.

RC522	Arduino Nano	Función
SDA / SS	D10	Selección SPI
SCK	D13	Reloj SPI
MOSI	D11	Datos hacia RC522
MISO	D12	Datos hacia Arduino
RST	D9	Reinicio del módulo
3.3 V	3.3 V	Alimentación
GND	GND	Tierra común

El módulo RC522 trabaja normalmente a 3.3 V. No debe alimentarse directamente con 5 V si el módulo no incluye regulación compatible. Se recomienda verificar la placa utilizada antes de energizar.

8.7.3. LEDs direccionables WS2812B

Los LEDs WS2812B integran un LED RGB y un controlador interno. Esto permite controlar muchos LEDs usando una sola línea de datos. Cada LED recibe información de color y transmite el resto de la cadena al siguiente LED [3].

Para facilitar el control desde Arduino, se utilizará la librería Adafruit NeoPixel [4]. Esta librería permite definir colores, brillo y efectos de animación.

8.7.4. Módulo DFPlayer Mini

El DFPlayer Mini es un módulo reproductor MP3 compacto que permite reproducir archivos de audio almacenados en una memoria microSD. Puede controlarse mediante comunicación serial

desde Arduino. DFRobot proporciona documentación y librería para controlar reproducción, volumen y selección de pistas [5, 7].

Los archivos deben guardarse en la microSD con nombres y carpetas compatibles con la librería utilizada. Una organización sencilla consiste en nombrar archivos como 0001.mp3, 0002.mp3, 0003.mp3, etc.

8.8. Diseño electrónico

Tabla 8.3: Asignación sugerida de pines para el cinturón interactivo.

Pin Arduino Nano	Conexión	Función
D2	RX DFPlayer	Comunicación serial software
D3	TX DFPlayer	Comunicación serial software
D6	Datos WS2812B	Control de LEDs
D9	RST RC522	Reinicio lector RFID
D10	SDA / SS RC522	Selección SPI
D11	MOSI RC522	Datos SPI
D12	MISO RC522	Datos SPI
D13	SCK RC522	Reloj SPI
5 V	LEDs DFPlayer	Alimentación
3.3 V	RC522	Alimentación RFID
GND	Todos los módulos	Tierra común

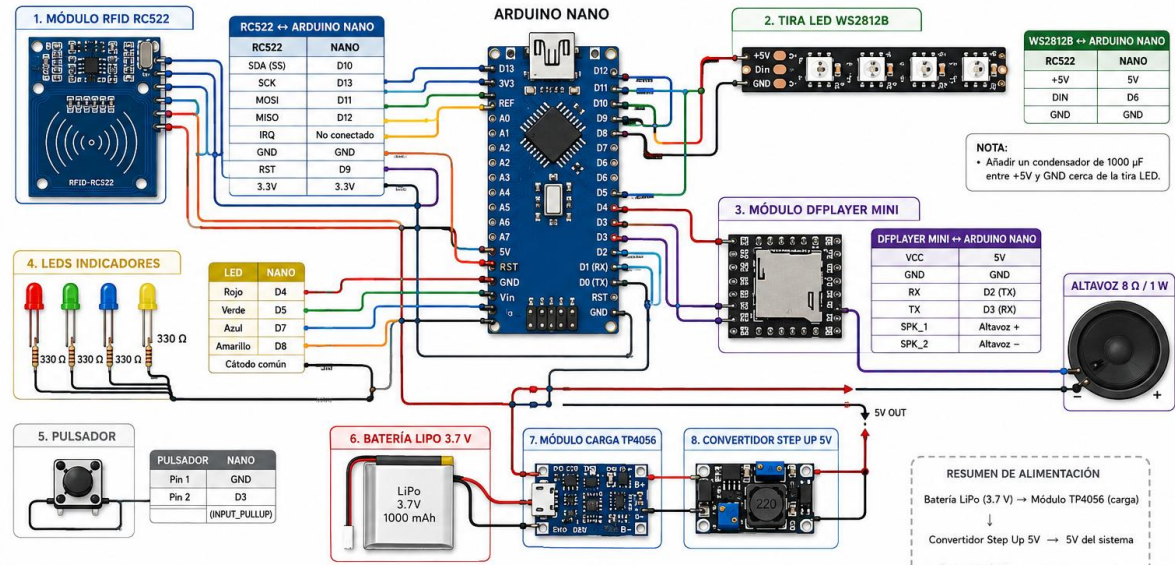


Figura 8.4: Diagrama de conexiones del cinturón interactivo RFID.

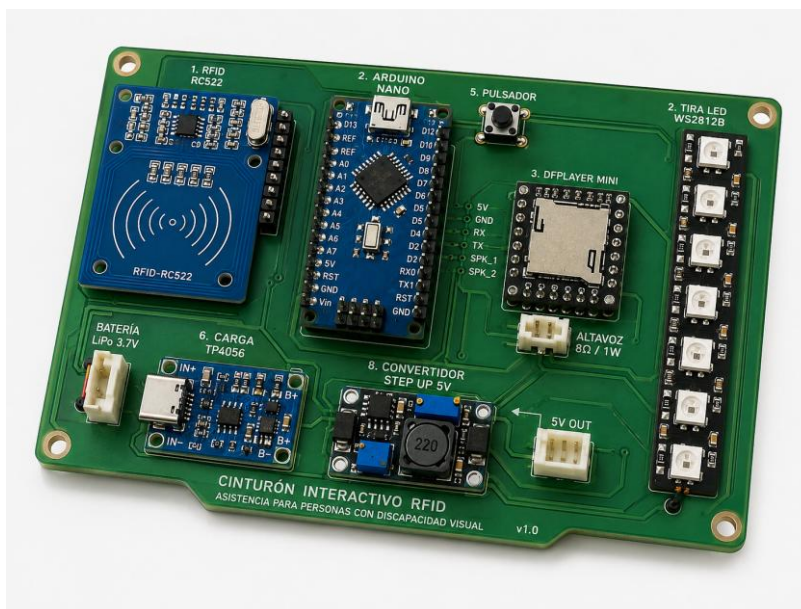


Figura 8.5: Montaje electrónico inicial del cinturón interactivo.

8.9. Diseño de interacción

El cinturón tendrá cuatro estados principales:

- **ESPERA:** efecto de respiración suave en los LEDs.
- **LECTURA:** detección de una tarjeta RFID.
- **ACTIVACIÓN:** reproducción de sonido y secuencia luminosa.
- **ERROR:** tarjeta no reconocida o lectura no autorizada.

Cada tarjeta puede asociarse con un modo:

Tabla 8.4: Ejemplo de modos asociados a tarjetas RFID.

Modo	Efecto LED	Archivo de audio
Modo 1	Barrido azul	0001.mp3
Modo 2	Parpadeo rojo	0002.mp3
Modo 3	Arco iris	0003.mp3
Modo error	Parpadeo blanco	0004.mp3

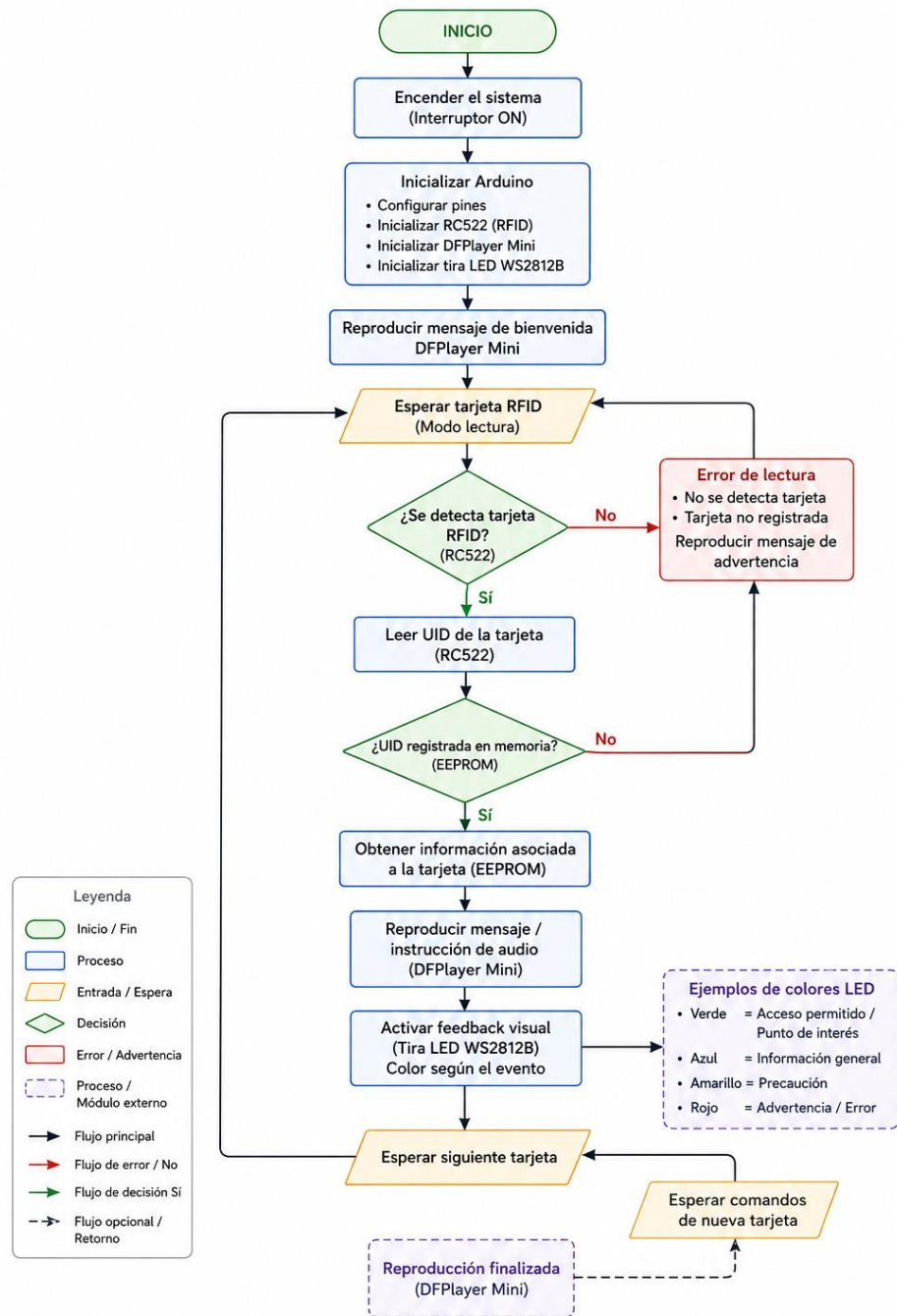


Figura 8.6: Máquina de estados del cinturón interactivo RFID.

8.10. Preparación de la memoria microSD

Para utilizar el DFPlayer Mini se recomienda:

1. Formatear la memoria microSD en FAT32.
2. Crear los archivos de audio en formato MP3.

3. Nombrar los archivos como 0001.mp3, 0002.mp3, 0003.mp3.
4. Insertar la memoria en el DFPlayer Mini con el sistema apagado.
5. Probar primero la reproducción de audio antes de integrar RFID y LEDs.

Nombre	Fecha de modificación	Tamaño	Clase
1.mp3	hoy 12:02 p.m.	15 KB	Audio MP3
2.mp3	hoy 12:04 p.m.	111 KB	Audio MP3

Figura 8.7: Organización sugerida de archivos de audio para el DFPlayer Mini.

8.11. Código del proyecto

El siguiente programa implementa lectura RFID, reconocimiento de UID, efectos LED básicos y reproducción de audio mediante DFPlayer Mini. Para compilarlo se requieren las librerías MFRC522, Adafruit_NeoPixel y DFRobotDFPlayerMini [8].

```
#include <SPI.h>
#include <MFRC522.h>
#include <Adafruit_NeoPixel.h>
#include <SoftwareSerial.h>
#include <DFRobotDFPlayerMini.h>

// =====
// PINES
// =====
const byte PIN_RFID_SS = 10;
const byte PIN_RFID_RST = 9;

const byte PIN_LEDS = 6;
const byte NUM_LEDS = 16;

const byte PIN_DF_RX = 2;
const byte PIN_DF_TX = 3;

// =====
// OBJETOS
// =====
MFRC522 rfid(PIN_RFID_SS, PIN_RFID_RST);
Adafruit_NeoPixel leds(NUM_LEDS, PIN_LEDS, NEO_GRB + NEO_KHZ800);

SoftwareSerial mp3Serial(PIN_DF_RX, PIN_DF_TX);
DFRobotDFPlayerMini mp3;

// =====
// UIDS AUTORIZADOS
// Reemplazar con los UID reales obtenidos en la prueba
// =====
```

```

byte uidModo1[4] = {0xDE, 0xAD, 0xBE, 0xEF};
byte uidModo2[4] = {0x12, 0x34, 0x56, 0x78};
byte uidModo3[4] = {0xA1, 0xB2, 0xC3, 0xD4};

// =====
// ESTADOS
// =====
enum EstadoCinturon
{
    ESPERA,
    ELECTURA,
    ACTIVACION,
    ERROR_RFID
};

EstadoCinturon estadoActual = ESPERA;

unsigned long tiempoEstado = 0;

// =====
// FUNCIONES LED
// =====
void apagarLeds()
{
    for (byte i = 0; i < NUM_LEDS; i++)
    {
        leds.setPixelColor(i, 0, 0, 0);
    }
    leds.show();
}

void colorCompleto(byte r, byte g, byte b)
{
    for (byte i = 0; i < NUM_LEDS; i++)
    {
        leds.setPixelColor(i, leds.Color(r, g, b));
    }
    leds.show();
}

void efectoBarrido(byte r, byte g, byte b, int espera)
{
    apagarLeds();

    for (byte i = 0; i < NUM_LEDS; i++)
    {
        leds.setPixelColor(i, leds.Color(r, g, b));
    }
}

```

```

    leds.show();
    delay(espera);
}
}

void efectoParpadeo(byte r, byte g, byte b, byte repeticiones)
{
    for (byte i = 0; i < repeticiones; i++)
    {
        colorCompleto(r, g, b);
        delay(120);
        apagarLeds();
        delay(120);
    }
}

void efectoEspera()
{
    static unsigned long t = 0;
    static byte brillo = 0;
    static int direccion = 1;

    if (millis() - t >= 20)
    {
        t = millis();

        brillo += direccion;

        if (brillo >= 80) direccion = -1;
        if (brillo <= 5) direccion = 1;

        colorCompleto(0, 0, brillo);
    }
}

// =====
// FUNCIONES RFID
// =====
bool compararUID(byte *uidLeido, byte *uidGuardado, byte longitud)
{
    for (byte i = 0; i < longitud; i++)
    {
        if (uidLeido[i] != uidGuardado[i])
        {
            return false;
        }
    }
}

```

```

    return true;
}

void imprimirUID()
{
    Serial.print("UID detectado: ");

    for (byte i = 0; i < rfid.uid.size; i++)
    {
        Serial.print(rfid.uid.uidByte[i], HEX);
        Serial.print(" ");
    }

    Serial.println();
}

int identificarModo()
{
    if (rfid.uid.size < 4)
    {
        return 0;
    }

    if (compararUID(rfid.uid.uidByte, uidModo1, 4)) return 1;
    if (compararUID(rfid.uid.uidByte, uidModo2, 4)) return 2;
    if (compararUID(rfid.uid.uidByte, uidModo3, 4)) return 3;

    return 0;
}

// =====
// MAQUINA DE ESTADOS
// =====
void cambiarEstado(EstadoCinturon nuevoEstado)
{
    estadoActual = nuevoEstado;
    tiempoEstado = millis();
}

void ejecutarEspera()
{
    efectoEspera();

    if (rfid.PICC_IsNewCardPresent() && rfid.PICC_ReadCardSerial())
    {
        imprimirUID();
    }
}

```



```

    cambiarEstado(LECTURA);
}
}

void ejecutarLectura()
{
    int modo = identificarModo();

    if (modo == 1)
    {
        mp3.play(1);
        efectoBarrido(0, 0, 255, 45);
        cambiarEstado(ACTIVACION);
    }
    else if (modo == 2)
    {
        mp3.play(2);
        efectoParpadeo(255, 0, 0, 5);
        cambiarEstado(ACTIVACION);
    }
    else if (modo == 3)
    {
        mp3.play(3);
        efectoBarrido(0, 255, 80, 35);
        cambiarEstado(ACTIVACION);
    }
    else
    {
        mp3.play(4);
        cambiarEstado(ERROR_RFID);
    }

    rfid.PICC_HaltA();
    rfid.PCD_StopCrypto1();
}

void ejecutarActivacion()
{
    colorCompleto(0, 80, 255);

    if (millis() - tiempoEstado >= 2000)
    {
        apagarLeds();
        cambiarEstado(ESPERA);
    }
}

```

```

void ejecutarError()
{
    efectoParpadeo(255, 255, 255, 3);
    cambiarEstado(ESPERA);
}

// =====
// SETUP Y LOOP
// =====

void setup()
{
    Serial.begin(115200);

    SPI.begin();
    rfid.PCD_Init();

    leds.begin();
    leds.setBrightness(80);
    apagarLeds();

    mp3Serial.begin(9600);
    if (mp3.begin(mp3Serial))
    {
        mp3.volume(22);
        Serial.println("DFPlayer iniciado");
    }
    else
    {
        Serial.println("Error al iniciar DFPlayer");
    }

    Serial.println("Cinturon RFID iniciado");
    cambiarEstado(ESPERA);
}

void loop()
{
    switch (estadoActual)
    {
        case ESPERA:
            ejecutarEspera();
            break;

        case LECTURA:
            ejecutarLectura();
            break;
    }
}

```

```
case ACTIVACION:
    ejecutarActivacion();
    break;

case ERROR_RFID:
    ejecutarError();
    break;

}

}
```

Los UID incluidos en el código son ejemplos. Para que el sistema funcione, primero se debe leer el UID real de cada tarjeta desde el monitor serial y reemplazar los valores del programa.

8.12. Lectura de UID de tarjetas

Antes de programar modos definitivos, se debe identificar el UID real de cada tarjeta o llavero. El programa principal imprime el UID detectado en el monitor serial. El estudiante debe acercarse a cada tarjeta, copiar los valores y reemplazarlos en los arreglos correspondientes.

Tabla 8.5: Formato sugerido para registro de tarjetas RFID.

Tarjeta	UID leído	Modo asignado
Tarjeta 1		Modo 1
Tarjeta 2		Modo 2
Tarjeta 3		Modo 3
Tarjeta no autorizada		Error

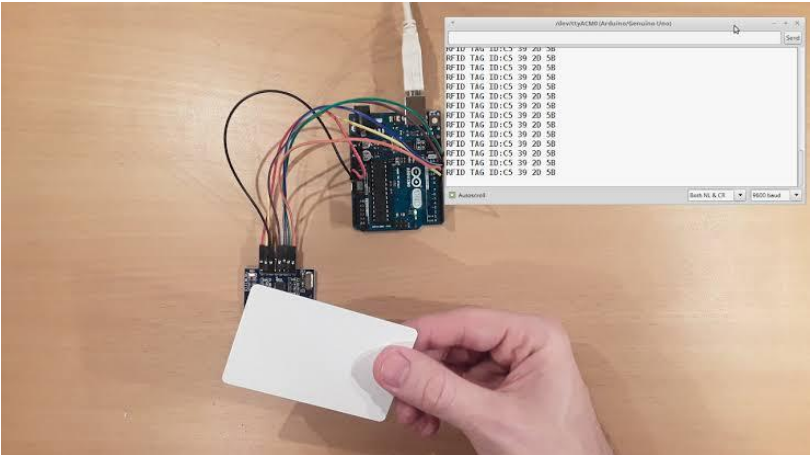


Figura 8.8: Lectura del UID de una tarjeta RFID para asignación de modos.

8.13. Diseño mecánico y fabricación

El cinturón puede construirse inicialmente sobre una base acrílica, cartón prensado, MDF delgado o impresión 3D. Para una versión más estética, se recomienda diseñar una carcasa en CAD que incluya:

- Alojamiento frontal para el lector RFID.
- Difusor translúcido para LEDs.
- Espacio para Arduino Nano.
- Compartimento para batería.
- Rejilla para salida de sonido del parlante.
- Canal interno para cableado.
- Sistema de sujeción al cinturón.

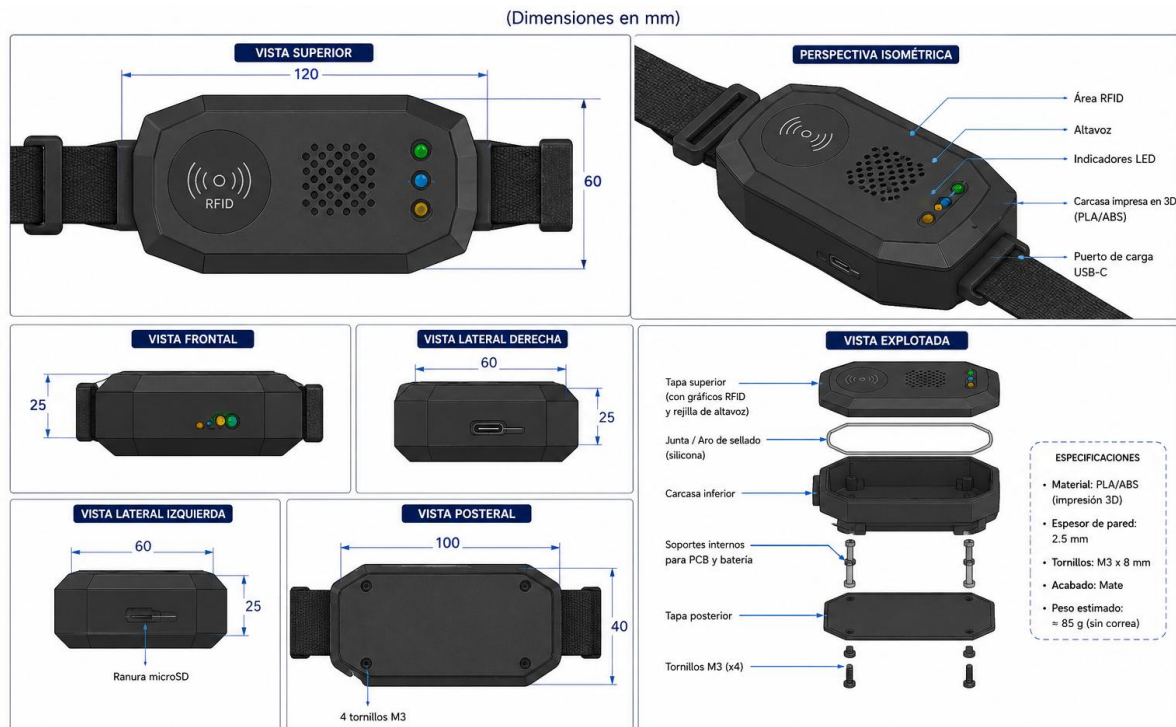


Figura 8.9: Diseño mecánico sugerido para el cinturón interactivo.

8.14. Pruebas experimentales

8.14.1. Prueba 1: lector RFID

Verificar que el monitor serial muestre correctamente el UID de cada tarjeta.

8.14.2. Prueba 2: LEDs WS2812B

Ejecutar efectos básicos de color completo, barrido y parpadeo sin conectar todavía el DFPlayer.

8.14.3. Prueba 3: DFPlayer Mini

Reproducir manualmente los archivos 0001.mp3, 0002.mp3 y 0003.mp3 para confirmar que la memoria microSD, parlante y módulo funcionan correctamente.

8.14.4. Prueba 4: integración completa

Acercar cada tarjeta y verificar que se active el efecto visual y sonoro correspondiente.

Tabla 8.6: Registro sugerido de pruebas del cinturón interactivo.

Prueba	Entrada	Resultado esperado	Resultado obtenido
RFID	Tarjeta 1	UID correcto	
LEDs	Modo 1	Barrido azul	
Audio	Archivo 0001	Reproducción correcta	
Integración	Tarjeta 2	Luz y sonido modo 2	
Error	Tarjeta no registrada	Efecto de error	

A. ENSAMBLAJE DEL DISPOSITIVO

1 Preparar las piezas impresas en 3D



2 Instalar los módulos en los soportes internos



3 Conectar antena RFID y altavoz



4 Conectar la tira LED WS2812B y demás cables



5 Colocar la junta de sellado (silicona)



6 Cerrar la carcasa con los tornillos M3 (x4)



Notas importantes:

- Verificar que todos los cables queden asegurados y no ejerzan presión sobre la tapa.
- Asegurar la batería con espuma adhesiva o bridas para evitar movimientos.

Indicadores LED (colores de referencia)

Verde	Acceso permitido / Punto de interés	Azul	Información general	Amarillo	Precaución	Rojo	Advertencia / Error
-------	-------------------------------------	------	---------------------	----------	------------	------	---------------------

B. USO DEL CINTURÓN

1. Colocación



Ajustar la correa a la cintura de forma cómoda y segura.

2. Presentación de tarjeta RFID



Acercar la tarjeta RFID al área designada (2-4 cm) para su lectura.

3. Respuesta del sistema

Color LED	Significado	Respuesta de audio
Verde	Acceso permitido / Punto de interés	Reproduce mensaje o instrucción asociada
Azul	Información general	Reproduce información general del punto
Amarillo	Precaución	Reproduce advertencia de precaución
Rojo	Advertencia / Error	Reproduce mensaje de error o prohibición

El audio se reproduce automáticamente a través del DFPlayer Mini y el altavoz integrado.

4. Uso continuo



El sistema permanece en modo lectura, listo para una nueva tarjeta.

Figura 8.10: Prueba funcional del cinturón interactivo RFID.

8.15. Errores frecuentes

Durante el desarrollo pueden presentarse los siguientes errores:

- Alimentar el RC522 con 5 V cuando el módulo requiere 3.3 V.
- Confundir las líneas MOSI y MISO de SPI.
- No conectar tierra común entre módulos.

- Usar una fuente insuficiente para la tira WS2812B.
- Nombrar incorrectamente los archivos MP3 en la microSD.
- Conectar RX y TX del DFPlayer sin cruzarlos correctamente.
- No copiar el UID real de la tarjeta en el código.
- Colocar el lector RFID detrás de un material que reduce demasiado el alcance.

8.16. Proyecto de extensión

Como extensión, se propone agregar una aplicación móvil mediante Bluetooth para seleccionar modos sin tarjeta RFID, modificar colores, ajustar brillo o elegir sonidos.

También se puede migrar el proyecto a ESP32 para incorporar control WiFi, servidor web o actualización inalámbrica de parámetros.

Otra extensión interesante consiste en registrar el número de activaciones de cada tarjeta en EEPROM, convirtiendo el cinturón en un sistema de perfiles de usuario.



Figura 8.11: Extensión del cinturón interactivo mediante aplicación móvil.

Actividades de aprendizaje

1. Lea el UID de tres tarjetas RFID diferentes y regístrelas en una tabla.
2. Modifique el color asociado a cada tarjeta.
3. Agregue un cuarto modo de activación con un nuevo archivo MP3.
4. Diseñe una carcasa para el cinturón usando software CAD.
5. Agregue un botón físico para repetir el último efecto activado.
6. Proponga una versión del proyecto basada en ESP32 y control WiFi.

Preguntas de reflexión

1. ¿Qué ventajas ofrece RFID frente a un simple pulsador?
2. ¿Por qué es importante separar el diseño electrónico del diseño de interacción?
3. ¿Qué problemas pueden aparecer al alimentar muchos LEDs WS2812B desde una fuente pequeña?
4. ¿Cómo se podría mejorar la seguridad eléctrica de un wearable alimentado por batería?
5. ¿Qué aplicaciones educativas podría tener un sistema de identificación RFID?

Referencias

- [1] NXP Semiconductors, “MFRC522 Standard performance MIFARE and NTAG frontend”, Product data sheet. [En línea]. Disponible en: <https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf>. [Accedido: 24-jun-2026].
- [2] M. Balboa, “MFRC522 RFID Library for Arduino”, GitHub. [En línea]. Disponible en: <https://github.com/miguelbalboa/rfid>. [Accedido: 24-jun-2026].
- [3] Worldsemi, “WS2812B Intelligent control LED integrated light source”, Datasheet. [En línea]. Disponible en: <https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf>. [Accedido: 24-jun-2026].
- [4] Adafruit, “Adafruit NeoPixel Library”, GitHub. [En línea]. Disponible en: https://github.com/adafruit/Adafruit_NeoPixel. [Accedido: 24-jun-2026].
- [5] DFRobot, “DFPlayer Mini SKU:DFR0299”, DFRobot Wiki. [En línea]. Disponible en: https://wiki.dfrobot.com/DFPlayer_Mini_SKU_DFR0299. [Accedido: 24-jun2026].
- [6] Arduino, “SPI Library”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/learn/communication/spi>. [Accedido: 24-jun-2026].
- [7] Arduino, “SoftwareSerial Library”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/learn/built-in-libraries/software-serial>. [Accedido: 24-jun-2026].
- [8] M. Banzi and M. Shiloh, *Getting Started with Arduino*, 4th ed. Sebastopol, CA, USA: Maker Media, 2022.

Capítulo 9. Guantelete de Control por Gestos con ESP32

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender el funcionamiento de un wearable de control basado en sensores flexibles e inerciales.
- Medir la flexión de los dedos mediante divisores resistivos y entradas analógicas del ESP32.
- Utilizar un MPU6050 para estimar inclinación y movimientos básicos de la mano.
- Diseñar un conjunto de gestos para enviar comandos hacia robots, aplicaciones móviles o sistemas interactivos.
- Transmitir comandos mediante Bluetooth clásico usando ESP32.
- Implementar calibración básica de sensores y filtrado digital para mejorar la estabilidad del sistema.
- Proponer extensiones del guantelete para control de robots, domótica, prótesis educativas o exhibiciones STEM.

9.1. Introducción

Los wearables tecnológicos son dispositivos electrónicos diseñados para ser utilizados sobre el cuerpo humano. Estos sistemas pueden medir variables fisiológicas, reconocer movimiento, controlar dispositivos externos, generar retroalimentación háptica o funcionar como interfaces de interacción. En educación STEM, los wearables permiten combinar electrónica, programación, diseño mecánico, comunicación inalámbrica y creatividad en un solo proyecto.

En este capítulo se desarrolla un guantelete de control por gestos basado en ESP32, sensores flexibles, MPU6050 y comunicación Bluetooth. El objetivo del proyecto es construir un dispositivo portátil capaz de reconocer posiciones simples de los dedos y movimientos de la mano para enviar comandos hacia un robot, una aplicación móvil o un sistema interactivo.

El guantelete puede utilizarse para controlar un robot móvil, activar modos de un sistema de luces, manipular un brazo robótico educativo, enviar comandos a una aplicación desarrollada en App Inventor o servir como demostrador tecnológico en ferias de robótica. A diferencia de un control remoto convencional, el guantelete transforma los movimientos naturales de la mano en instrucciones digitales [8].

El proyecto integra dos tipos principales de sensores. Los sensores flexibles permiten detectar la curvatura de los dedos mediante cambios de resistencia, mientras que el MPU6050 proporciona información inercial sobre orientación y movimiento. El ESP32 actúa como unidad central de procesamiento y comunicación inalámbrica. La documentación de Arduino-ESP32

incluye soporte para Bluetooth clásico mediante la clase BluetoothSerial, que permite comunicación tipo puerto serial inalámbrico en placas ESP32 compatibles [1].



Figura 9.1: Guantelete de control por gestos basado en ESP32 y sensores flexibles.

9.2. Planteamiento del problema

Los robots y sistemas interactivos suelen controlarse mediante joysticks, botones, controles RC o aplicaciones móviles. Aunque estos métodos son funcionales, no siempre resultan intuitivos para demostraciones educativas o experiencias de interacción natural. Un guantelete de control permite explorar una alternativa más inmersiva, donde el usuario genera comandos mediante flexión de dedos e inclinación de la mano.

El desarrollo del guantelete plantea varios desafíos:

- Medir señales analógicas variables producidas por sensores flexibles.
- Calibrar cada dedo, ya que los sensores no siempre entregan el mismo rango.
- Evitar falsas detecciones por ruido o movimientos pequeños.
- Combinar flexión de dedos con inclinación de la mano.
- Enviar comandos de manera clara y confiable.
- Mantener el sistema cómodo, portable y seguro.

Un wearable de control no solo debe detectar movimiento. Debe traducir gestos humanos en comandos digitales claros, repetibles y fáciles de interpretar por otro sistema.

9.3. Descripción general del proyecto

El guantelete propuesto utiliza tres sensores flexibles para medir los dedos índice, medio y pulgar. Además, incorpora un MPU6050 para detectar inclinación de la mano. El ESP32 procesa estas señales y envía comandos por Bluetooth [7].

Los comandos básicos propuestos son:

- Mano abierta: detener.
- Mano inclinada hacia adelante: avanzar.
- Mano inclinada hacia atrás: retroceder.
- Inclinación hacia la izquierda: girar izquierda.
- Inclinación hacia la derecha: girar derecha.
- Puño cerrado: activar modo especial o agarre.

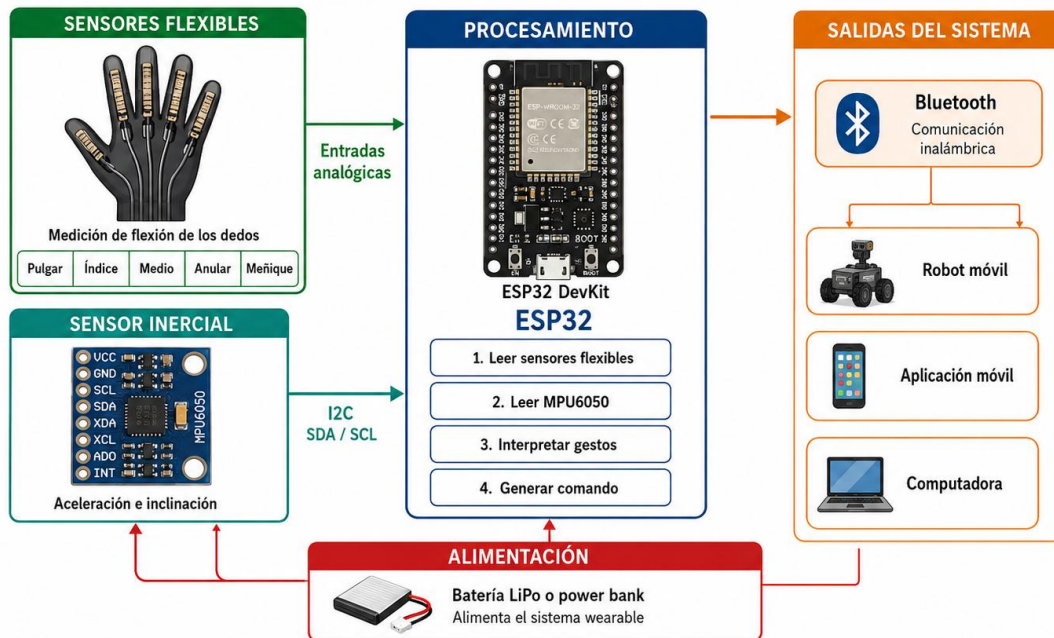


Figura 9.2: Arquitectura general del guantelete de control por gestos.

9.4. Aplicación STEM

Este proyecto integra los componentes STEM de la siguiente forma:

- **Ciencia:** análisis de movimiento, inclinación, resistencia variable y aceleración.
- **Tecnología:** uso de ESP32, Bluetooth, sensores flexibles, IMU y comunicación I2C.
- **Ingeniería:** diseño de una interfaz wearable usable, segura y adaptable.
- **Matemáticas:** normalización de señales, umbrales, filtrado, ángulos e interpretación de gestos.

9.5. Objetivos del proyecto

9.5.1. Objetivo general

Diseñar e implementar un guantelete de control por gestos capaz de reconocer flexión de dedos e inclinación de la mano para enviar comandos inalámbricos mediante ESP32 y Bluetooth.

9.5.2. *Objetivos específicos*

- 1. Medir sensores flexibles mediante divisores resistivos.
- 2. Leer aceleración del MPU6050 mediante comunicación I2C.
- 3. Calibrar valores mínimos y máximos de flexión.
- 4. Implementar filtrado digital para estabilizar las lecturas.
- 5. Definir gestos básicos y asociarlos con comandos.
- 6. Enviar comandos por Bluetooth clásico.
- 7. Proponer una interfaz de recepción en robot o aplicación móvil.

9.6. **Materiales y herramientas**

Tabla 9.1: *Materiales principales para el guantelete de control por gestos.*

Elemento	Cantidad	Función
ESP32 DevKit	1	Procesamiento y comunicación Bluetooth
Sensores flexibles	3 a 5	Medición de flexión de dedos
Resistencias 22K Ω – 47 K Ω	3 a 5	Divisores de tensión
MPU6050	1	Medición de aceleración e inclinación
Guante de tela	1	Soporte wearable
Cables flexibles	Varios	Interconexión cómoda
Batería USB o LiPo con regulador	1	Alimentación portátil
Cinta, velcro o soporte impreso	Varios	Fijación de sensores
Computadora con Arduino IDE	1	Programación y monitoreo
Robot o app receptora	1	Sistema controlado

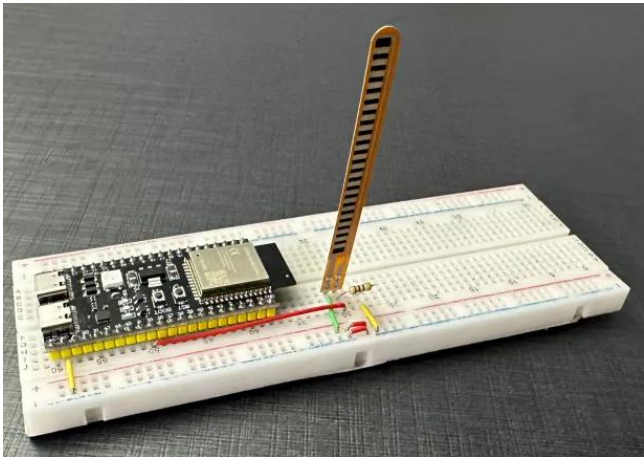


Figura 9.3: *Materiales utilizados para construir el guantelete de control por gestos.*

9.7. Fundamento teórico

9.7.1. Sensores flexibles

Un sensor flexible es un componente resistivo cuya resistencia cambia al doblarse. En general, al aumentar la flexión, cambia la resistencia del elemento sensor. Para leer este cambio con un microcontrolador se utiliza un divisor de voltaje. SparkFun recomienda conectar el sensor flexible junto con una resistencia fija para crear una señal analógica medible por Arduino u otros microcontroladores [2].

El divisor resistivo puede representarse como:

$$V_{out} = V_{cc} \left(\frac{R_f}{R_{flex} + R_f} \right)$$

donde:

- V_{out} es el voltaje leído por el ADC.
- R_{flex} es la resistencia variable del sensor flexible.
- R_f es la resistencia fija del divisor.
- V_{cc} es el voltaje de alimentación.

El valor de la resistencia fija debe elegirse según el rango real del sensor. En muchos casos se utilizan resistencias entre 22 K Ω y 47 K Ω , pero la mejor elección debe determinarse experimentalmente.

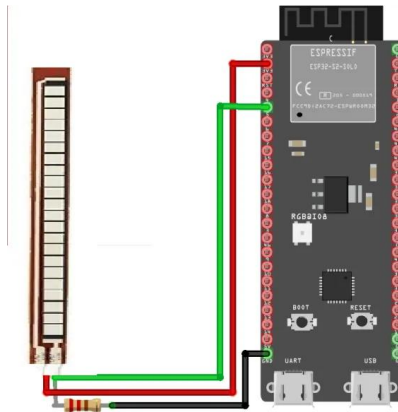


Figura 9.4: Divisor resistivo para lectura de un sensor flexible.

9.7.2. MPU6050

El MPU6050 integra acelerómetro de tres ejes y giroscopio de tres ejes. La ficha técnica del dispositivo lo describe como una unidad de seguimiento de movimiento de seis ejes con comunicación I2C [1, 4]. En este proyecto se utiliza principalmente el acelerómetro para estimar inclinación simple de la mano [3, 6].

La aceleración resultante puede calcularse mediante:

$$a_R = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

Para estimar inclinación aproximada se pueden usar relaciones trigonométricas simples. Por ejemplo, una estimación del ángulo de inclinación frontal puede expresarse como:

$$\theta_x = \tan^{-1} \left(\frac{a_x}{\sqrt{a_y^2 + a_z^2}} \right)$$

9.7.3. Bluetooth clásico en ESP32

El ESP32 permite comunicación Bluetooth clásico en muchas placas basadas en el chip ESP32 original. La documentación de Arduino-ESP32 muestra el uso de `BluetoothSerial` para crear un puente entre comunicación serial USB y Bluetooth clásico SPP [1]. Esta característica permite enviar comandos de texto como si se tratara de un puerto serial inalámbrico.

MIT App Inventor incluye el componente `BluetoothClient`, que permite conectar una app Android a otros dispositivos usando Bluetooth mediante el perfil de puerto serial SPP [5]. Esto facilita crear una app receptora para visualizar gestos o enviar comandos a otro sistema.

9.8. Diseño electrónico

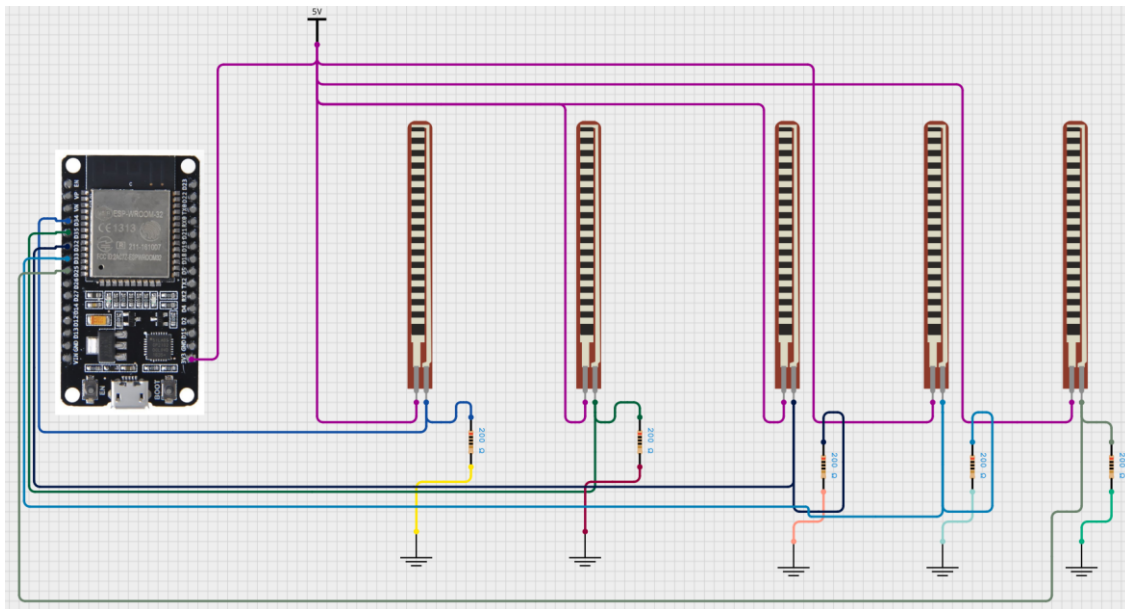


Figura 9.5: Diagrama de conexiones del guantelete de control por gestos.

Tabla 9.2: Asignación sugerida de pines para el guantelete.

Pin ESP32	Conexión	Función
GPIO 34	Flex índice	Entrada analógica
GPIO 35	Flex medio	Entrada analógica
GPIO 32	Flex pulgar	Entrada analógica
GPIO 21	SDA MPU6050	Comunicación I2C
GPIO 22	SCL MPU6050	Comunicación I2C
3.3 V	Flex y MPU6050	Alimentación lógica
GND	Todos los módulos	Tierra común
USB / batería	ESP32	Alimentación del sistema



Figura 9.6: Montaje físico del guantelete de control por gestos.

9.9. Diseño de gestos

El reconocimiento de gestos se basa en dos tipos de información:

- Flexión de dedos.
- Inclinación de la mano.

En una versión inicial se recomienda usar pocos gestos, claramente diferenciados.

Tabla 9.3: Gestos básicos propuestos para el guantelete.

Gesto	Condición	Comando enviado
Mano abierta	Dedos extendidos	STOP
Inclinación adelante	Mano inclinada hacia adelante	FWD
Inclinación atrás	Mano inclinada hacia atrás	BACK
Inclinación izquierda	Mano inclinada a la izquierda	LEFT
Inclinación derecha	Mano inclinada a la derecha	RIGHT
Puño cerrado	Dedos flexionados	GRIP

Es preferible iniciar con gestos simples y robustos. Un sistema con pocos gestos confiables es mejor que uno con muchos gestos difíciles de repetir.

9.10. Calibración de sensores flexibles

Cada sensor flexible puede entregar valores diferentes, incluso si pertenece al mismo modelo. Por ello, se debe calibrar el valor correspondiente a dedo extendido y dedo flexionado.

```
const byte FLEX_INDICE = 34;
const byte FLEX_MEDIO = 35;
const byte FLEX_PULGAR = 32;

void setup()
{
  Serial.begin(115200);
}

void loop()
{
  Serial.print("Indice:");
  Serial.print(analogRead(FLEX_INDICE));

  Serial.print(" Medio:");
  Serial.print(analogRead(FLEX_MEDIO));

  Serial.print(" Pulgar:");
  Serial.println(analogRead(FLEX_PULGAR));

  delay(100);
}
```

Se recomienda llenar una tabla de calibración:

Tabla 9.4: Formato sugerido para calibración de sensores flexibles.

Sensor	Dedo extendido	Dedo flexionado	Umbral elegido
Índice			
Medio			
Pulgar			

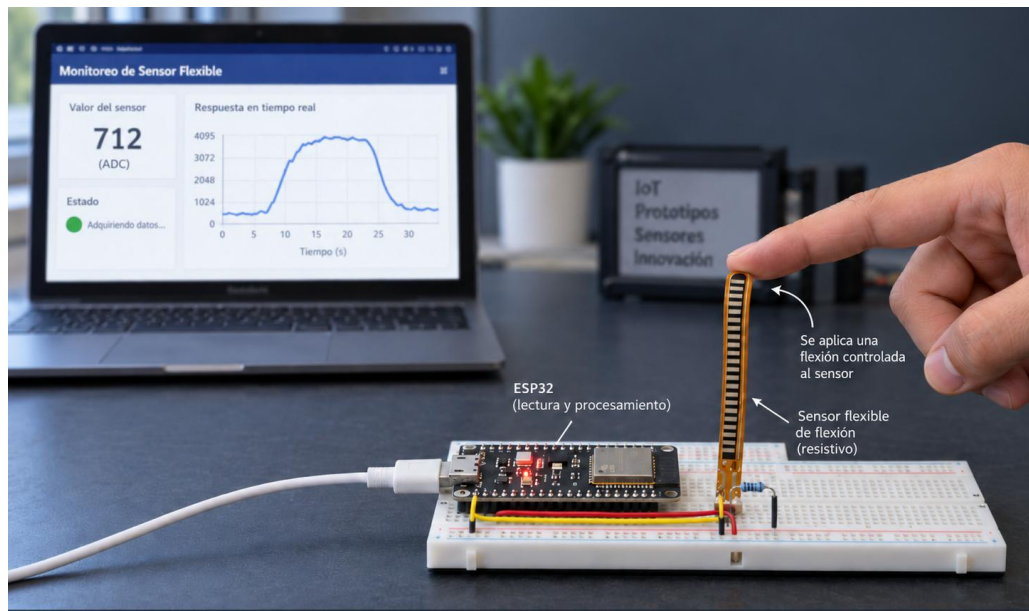


Figura 9.7: Calibración experimental de sensores flexibles.

9.11. Código del proyecto

El siguiente programa implementa lectura de sensores flexibles, lectura básica del MPU6050, detección de gestos y envío de comandos por Bluetooth clásico. Para simplificar el aprendizaje, la lectura del MPU6050 se realiza directamente con la librería Wire.

```
#include <Wire.h>
#include "BluetoothSerial.h"

// =====
// BLUETOOTH
// =====
BluetoothSerial SerialBT;

// =====
// PINES
// =====
const byte FLEX_INDICE = 34;
const byte FLEX_MEDIO = 35;
const byte FLEX_PULGAR = 32;

// MPU6050
const byte MPU_ADDR = 0x68;
const byte REG_PWR_MGMT_1 = 0x6B;
const byte REG_ACCEL_XOUT_H = 0x3B;

// =====
```



```

// PARAMETROS DE CALIBRACION
// Ajustar con valores reales del guantelete
// =====
int UMBRAL_INDICE = 2400;
int UMBRAL_MEDIO = 2400;
int UMBRAL_PULGAR = 2400;

const float SENSIBILIDAD_ACCEL = 16384.0;

// Filtro
const float ALPHA = 0.30;

// Variables filtradas
float flexIndiceF = 0;
float flexMedioF = 0;
float flexPulgarF = 0;

float ax, ay, az;
float anguloX, anguloY;

String ultimoComando = "";
unsigned long tiempoEnvio = 0;

// =====
// FUNCIONES MPU6050
// =====
void iniciarMPU6050()
{
    Wire.begin(21, 22);

    Wire.beginTransmission(MPU_ADDR);
    Wire.write(REG_PWR_MGMT_1);
    Wire.write(0);
    Wire.endTransmission(true);
}

int16_t leerWordMPU(byte registro)
5{
    Wire.beginTransmission(MPU_ADDR);
    Wire.write(registro);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU_ADDR, (byte)2, true);

    int16_t valor = Wire.read() << 8 | Wire.read();
6return valor;
}

```

```

void leerMPU6050()
{
    int16_t rawAx = leerWordMPU(REG_ACCEL_XOUT_H);
    int16_t rawAy = leerWordMPU(REG_ACCEL_XOUT_H + 2);
    int16_t rawAz = leerWordMPU(REG_ACCEL_XOUT_H + 4);

    ax = rawAx / SENSIBILIDAD_ACCEL;
    ay = rawAy / SENSIBILIDAD_ACCEL;
    az = rawAz / SENSIBILIDAD_ACCEL;

    anguloX = atan2(ax, sqrt(ay * ay + az * az)) * 180.0 / PI;
    anguloY = atan2(ay, sqrt(ax * ax + az * az)) * 180.0 / PI;
}

// =====
// FLEX SENSORS
// =====
void leerFlex()
{
    int indice = analogRead(FLEX_INDICE);
    int medio = analogRead(FLEX_MEDIO);
    int pulgar = analogRead(FLEX_PULGAR);

    flexIndiceF = ALPHA * indice + (1 - ALPHA) * flexIndiceF;
    flexMedioF = ALPHA * medio + (1 - ALPHA) * flexMedioF;
    flexPulgarF = ALPHA * pulgar + (1 - ALPHA) * flexPulgarF;
}

bool indiceFlexionado()
{
    return flexIndiceF > UMBRAL_INDICE;
}

bool medioFlexionado()
{
    return flexMedioF > UMBRAL_MEDIO;
}

bool pulgarFlexionado()
{
    return flexPulgarF > UMBRAL_PULGAR;
}

bool punoCerrado()
{
    return indiceFlexionado() && medioFlexionado() && pulgarFlexionado();
}

```

```

bool manoAbierta()
{
    return !indiceFlexionado() && !medioFlexionado() && !pulgarFlexionado();
}

// =====
// RECONOCIMIENTO DE GESTOS
// =====
String detectarComando()
{
    if (punoCerrado())
    {
        return "GRIP";
    }

    if (manoAbierta())
    {
        if (anguloX > 25) return "FWD";
        if (anguloX < -25) return "BACK";
        if (anguloY > 25) return "RIGHT";
        if (anguloY < -25) return "LEFT";

        return "STOP";
    }

    if (indiceFlexionado() && !medioFlexionado())
    {
        return "MODE1";
    }

    if (!indiceFlexionado() && medioFlexionado())
    {
        return "MODE2";
    }

    return "STOP";
}

// =====
// ENVIO DE COMANDOS
// =====
void enviarComando(String comando)
{
    if (millis() - tiempoEnvio >= 120)
    {
        tiempoEnvio = millis();
    }
}

```

```

    if (comando != ultimoComando)
    {
        ultimoComando = comando;
        SerialBT.println(comando);
        Serial.println(comando);
    }
}

void enviarTelemetria()
{
    static unsigned long t = 0;

    if (millis() - t >= 500)
    {
        t = millis();

        Serial.print("FI:");
        Serial.print(flexIndiceF);

        Serial.print(" FM:");
        Serial.print(flexMedioF);

        Serial.print(" FP:");
        Serial.print(flexPulgarF);

        Serial.print(" AX:");
        Serial.print(anguloX);

        Serial.print(" AY:");
        Serial.print(anguloY);

        Serial.print(" CMD:");
        Serial.println(ultimoComando);
    }
}

// =====
// SETUP Y LOOP
// =====

void setup()
{
    Serial.begin(115200);
    flexIndiceF = analogRead(FLEX_INDICE);
    flexMedioF = analogRead(FLEX_MEDIO);
    flexPulgarF = analogRead(FLEX_PULGAR);
}

```

```

iniciarMPU6050();

SerialBT.begin("Guantelete_ESP32");

Serial.println("Guantelete de control por gestos iniciado");
Serial.println("Bluetooth: Guantelete_ESP32");
}

void loop()
{
  leerFlex();
  leerMPU6050();

  String comando = detectarComando();
  enviarComando(comando);
  enviarTelemetria();
}

```

Los umbrales incluidos en el código son referenciales. Cada guantelete debe calibrarse individualmente, porque la lectura depende del sensor flexible, la resistencia fija, la posición del dedo, el montaje físico y la alimentación.

9.12. Protocolo de comandos

El guantelete envía comandos de texto por Bluetooth. Esto facilita la depuración, ya que los comandos pueden leerse desde una app, una computadora o un robot receptor.

Tabla 9.5: Comandos enviados por el guantelete.

Comando	Gesto asociado	Uso sugerido
STOP	Mano abierta sin inclinación	Detener robot
FWD	Mano abierta inclinada adelante	Avanzar
BACK	Mano abierta inclinada atrás	Retroceder
LEFT	Mano inclinada izquierda	Gira izquierda
RIGHT	Mano inclinada derecha	Gira derecha
GRIP	Puño cerrado	Activar garra o modo especial
MODE1	Índice flexionado	Activar modo 1
MODE2	Medio flexionado	Activar modo 2

9.13. Receptor básico para robot

El robot receptor puede usar otro ESP32 con Bluetooth clásico. El receptor interpreta comandos de texto y ejecuta acciones. A continuación, se presenta un ejemplo conceptual que puede adaptarse a un robot móvil.

```

#include "BluetoothSerial.h"

BluetoothSerial SerialBT;

String comando = "";

void setup()
{
  Serial.begin(115200);
  SerialBT.begin("Robot_Receptor");
  Serial.println("Robot receptor iniciado");
}

void loop()
{
  if (SerialBT.available())
  {
    comando = SerialBT.readStringUntil('\n');
    comando.trim();

    if (comando == "FWD")
    {
      Serial.println("Avanzar");
    }
    else if (comando == "BACK")
    {
      Serial.println("Retroceder");
    }
    else if (comando == "LEFT")
    {
      Serial.println("Girar izquierda");
    }
    else if (comando == "RIGHT")
    {
      Serial.println("Girar derecha");
    }
    else if (comando == "STOP")
    {
      Serial.println("Detener");
    }
    else if (comando == "GRIP")
    {
      Serial.println("Activar modo especial");
    }
  }
}

```



Figura 9.8: Control inalámbrico de un robot mediante guantelete de gestos.

9.14. Integración con App Inventor

Una alternativa educativa consiste en recibir los comandos del guantelete en una app desarrollada en MIT App Inventor. El componente BluetoothClient permite conectar dispositivos mediante Bluetooth clásico SPP [5]. La app puede mostrar el comando recibido, cambiar colores en pantalla o reenviar instrucciones a otro sistema.

Elementos sugeridos para la app:

- Selector de dispositivo Bluetooth.
- Etiqueta para comando recibido.
- Indicador visual de dirección.
- Botón de conexión.
- Registro de últimos comandos.

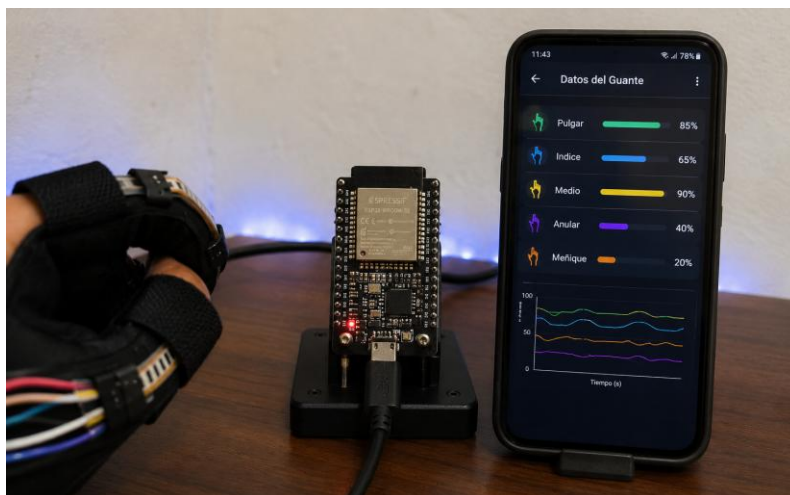


Figura 9.9: Aplicación móvil para monitoreo de comandos del guantelete.

9.15. Diseño mecánico y ergonomía

El guantelete debe ser cómodo, flexible y seguro. Los sensores no deben impedir el movimiento natural de la mano. Se recomienda:

- Fijar sensores flexibles sobre los dedos sin doblarlos excesivamente.
- Usar cables delgados y flexibles.
- Proteger soldaduras con termoencogible.
- Colocar el ESP32 sobre el dorso de la mano o antebrazo.
- Evitar bordes filosos o piezas rígidas incómodas.
- Permitir retirar la batería fácilmente.



Figura 9.10: Diseño mecánico sugerido para el guantelete de control.

9.16. Pruebas experimentales

9.16.1. Prueba 1: sensores flexibles

Registrar valores de cada dedo extendido y flexionado. Ajustar umbrales hasta obtener una detección estable.

9.16.2. Prueba 2: MPU6050

Inclinar la mano hacia adelante, atrás, izquierda y derecha. Verificar que los ángulos cambien en la dirección esperada.

9.16.3. Prueba 3: Bluetooth

Conectar el guantelete a una computadora, celular o ESP32 receptor. Verificar que los comandos lleguen correctamente.

9.16.4. Prueba 4: integración con robot

Enviar comandos hacia un robot móvil en baja velocidad. Verificar que cada gesto produzca la acción esperada.

Tabla 9.6: Registro sugerido de pruebas del guantelete.

Prueba	Gesto	Comando esperado	Resultado obtenido
Mano abierta	Sin inclinación	STOP	
Inclinación adelante	Mano abierta	FWD	
Inclinación izquierda	Mano abierta	LEFT	
Puño cerrado	Dedos flexionados	GRIP	
Índice flexionado	Índice doblado	MODE1	

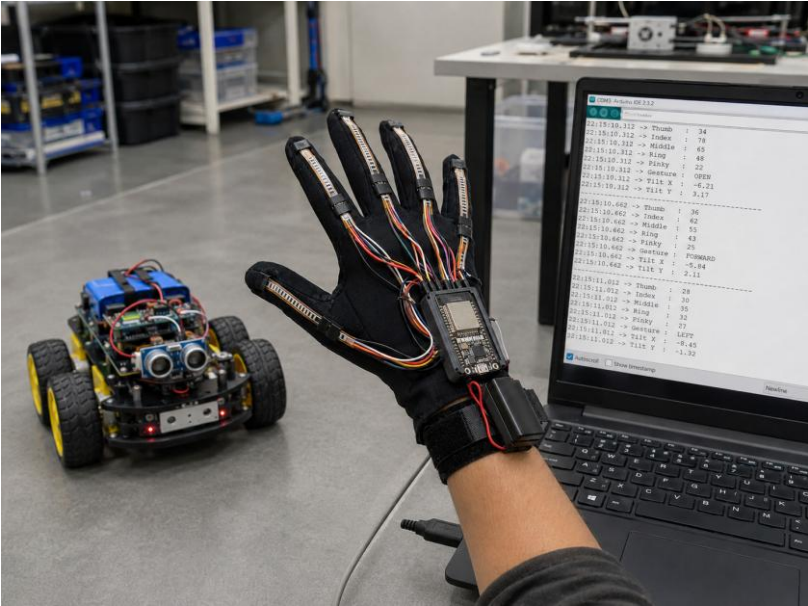


Figura 9.11: Validación experimental del guantelete de control por gestos.

9.17. Errores frecuentes

Durante el desarrollo pueden presentarse los siguientes errores:

- Conectar sensores flexibles sin divisor resistivo.
- Usar pines del ESP32 que no admiten entrada analógica.
- Definir umbrales sin calibración previa.
- Montar sensores demasiado tensos sobre los dedos.
- No compartir tierra entre módulos.
- Confundir los ejes del MPU6050.
- Usar una placa ESP32 sin soporte de Bluetooth clásico.
- Enviar comandos demasiado rápido y saturar el receptor.

9.18. Proyecto de extensión

Como extensión, se propone agregar retroalimentación háptica mediante motores vibradores ubicados en la muñeca. Estos actuadores podrían indicar conexión Bluetooth, comando enviado o alerta de batería baja.

Otra extensión consiste en entrenar gestos más complejos mediante aprendizaje automático simple, registrando datos de flexión e inclinación para clasificarlos posteriormente. También puede agregarse una pantalla OLED pequeña para mostrar el comando actual y el estado de conexión.



Figura 9.12: Extensiones posibles para el guantelete de control por gestos.

Actividades de aprendizaje

1. Calibre los sensores flexibles del guantelete y complete la tabla de valores.
2. Modifique los umbrales para mejorar la detección de puño cerrado.
3. Agregue un nuevo comando usando la flexión del pulgar.
4. Diseñe una app en App Inventor que muestre el comando recibido por Bluetooth.
5. Integre el guantelete con un robot móvil a baja velocidad.
6. Proponga una versión del guantelete con retroalimentación háptica.

Preguntas de reflexión

1. ¿Por qué los sensores flexibles requieren calibración individual?
2. ¿Qué ventaja ofrece combinar flexión de dedos con inclinación de la mano?
3. ¿Qué problemas podrían aparecer si se envían comandos Bluetooth demasiado rápido?

4. ¿Por qué la ergonomía es importante en el diseño de wearables?
5. ¿Cómo podría utilizarse este guantelete en una exhibición de robótica educativa?
6. ¿Qué mejoras serían necesarias para controlar un robot de forma más precisa?

Referencias

- [1] Espressif Systems, “Bluetooth”, Arduino ESP32 Documentation. [En línea]. Disponible en: <https://docs.espressif.com/projects/arduino-esp32/en/latest/api/bluetooth.html>. [Accedido: 24-jun-2026].
- [2] SparkFun, “Flex Sensor Hookup Guide”, SparkFun Learn. [En línea]. Disponible en: <https://learn.sparkfun.com/tutorials/flex-sensor-hookup-guide/all> [Accedido: 24-jun-2026].
- [3] TDK Product Center, “MPU-6050: Detailed Information”. [En línea]. Disponible en: https://product.tdk.com/en/search/sensor/motion-inertial/imu/info?part_no=MPU-6050. [Accedido: 24-jun-2026].
- [4] InvenSense, “MPU-6000 and MPU-6050 Product Specification”, Rev. 3.4. [En línea]. Disponible en: <https://cdn.sparkfun.com/datasheets/Components/General%20IC/PS-MPU-6000A.pdf>. [Accedido: 24-jun-2026].
- [5] MIT App Inventor, “Connectivity components: BluetoothClient”, MIT App Inventor Documentation. [En línea]. Disponible en: <https://ai2.appinventor.mit.edu/reference/components/connectivity.html>. [Accedido: 24-jun-2026].
- [6] Arduino, “Wire Library”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/learn/communication/wire>. [Accedido: 24-jun-2026].
- [7] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2017.
- [8] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed. Cham, Switzerland: Springer, 2016.

Capítulo 10. Medición del Centro de Masa con Celdas de Carga y HX711

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender la importancia del centro de masa en robots Minisumo, Sumo, seguidores de línea y robots de combate.
- Utilizar celdas de carga y módulos HX711 para medir distribución de peso.
- Calibrar celdas de carga mediante masas conocidas.
- Calcular la posición aproximada del centro de masa en dos dimensiones.
- Interpretar la relación entre distribución de peso, tracción, estabilidad y desempeño competitivo.
- Construir una plataforma experimental para comparar diferentes configuraciones mecánicas de un robot.
- Registrar datos por puerto serial para análisis en hoja de cálculo, MATLAB o Python.

10.1. Introducción

La distribución de masa es un factor determinante en el desempeño de un robot competitivo. En un robot Minisumo, influye directamente en la capacidad de empuje, la adherencia de las ruedas y la estabilidad durante el ataque. En un seguidor de línea de alta velocidad, afecta la respuesta en curvas, el frenado y la tendencia a abrirse o derrapar. En robots de combate, la ubicación de baterías, motores y estructuras puede modificar la estabilidad ante impactos.

A pesar de su importancia, el centro de masa suele ajustarse de manera empírica. El diseñador mueve la batería, cambia la posición de motores, agrega peso frontal o modifica la altura del chasis hasta observar una mejora. Este proceso es válido, pero puede volverse más eficiente si se acompaña de mediciones. Medir permite comparar configuraciones, justificar decisiones y documentar resultados de manera objetiva.

Este capítulo desarrolla una plataforma de medición del centro de masa utilizando celdas de carga y módulos HX711. El sistema permite colocar un robot sobre cuatro puntos de apoyo, medir la fuerza en cada esquina y estimar la posición aproximada del centro de masa en el plano horizontal.

El proyecto tiene una aplicación directa en robótica competitiva. Por ejemplo, un Minisumo puede requerir mayor carga sobre las ruedas motrices para mejorar tracción; un seguidor de línea puede necesitar un centro de masa bajo y bien distribuido; un robot de combate puede requerir ubicar componentes críticos de forma que no afecten la estabilidad [5].

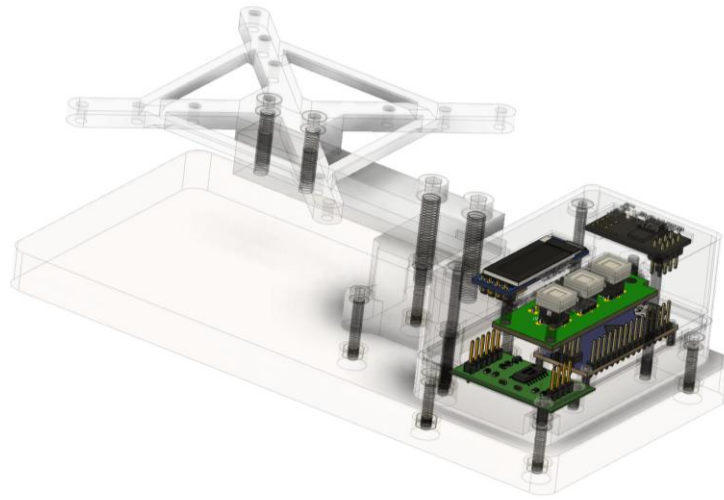


Figura 10.1: Plataforma experimental para medición de centro de masa en robots competitivos.

10.2. Planteamiento del problema

En la práctica, dos robots con motores similares, baterías similares y algoritmos similares pueden comportarse de manera muy diferente debido a su distribución de peso. Un robot con demasiado peso en la parte posterior puede perder capacidad de ataque frontal. Uno con exceso de peso delantero puede levantar ligeramente sus ruedas motrices durante ciertas maniobras. Un robot con masa lateralmente desbalanceada puede girar de manera desigual.

Algunos problemas asociados a una mala distribución de masa son:

Pérdida de tracción en una o más ruedas. Tendencia a levantarse durante aceleraciones bruscas. Giros asimétricos. Inestabilidad durante impactos. Dificultad para mantener la línea en curvas cerradas. Desgaste desigual de llantas. Comportamiento impredecible al cambiar baterías o accesorios.

Medir el centro de masa permite convertir un ajuste mecánico intuitivo en un procedimiento experimental. Esto ayuda a comparar configuraciones y a tomar decisiones basadas en datos.

10.3. Descripción general del proyecto

La plataforma propuesta utiliza cuatro celdas de carga ubicadas en las esquinas de una base rectangular. Cada celda mide la fuerza aplicada en su punto de apoyo. Con estas cuatro mediciones se calcula el peso total del robot y la posición aproximada de su centro de masa en los ejes x e y .

El sistema puede implementarse con cuatro módulos HX711, uno por cada celda de carga, conectados a un Arduino Nano. También puede adaptarse a ESP32 si se desea transmitir datos por WiFi o Bluetooth en una versión posterior [4].

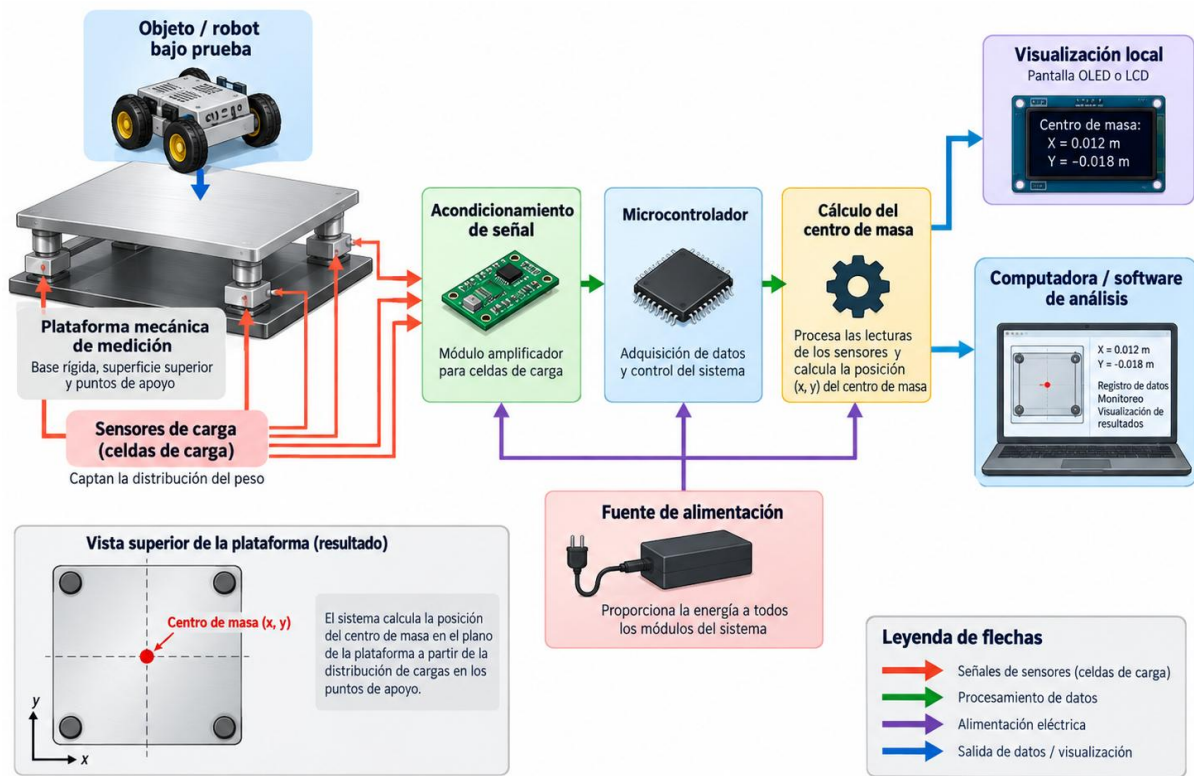


Figura 10.2: Arquitectura general de la plataforma de medición de centro de masa.

10.4. Aplicación STEM

Este proyecto integra las áreas STEM de la siguiente forma:

- **Ciencia:** análisis de fuerzas, peso, equilibrio estático y estabilidad.
- **Tecnología:** uso de celdas de carga, amplificadores HX711 y microcontroladores.
- **Ingeniería:** diseño de una plataforma de medición para optimizar robots reales.
- **Matemáticas:** cálculo de sumatorias de fuerzas, momentos y coordenadas del centro de masa.

10.5. Objetivos del proyecto

10.5.1. Objetivo general

Diseñar e implementar una plataforma experimental para medir el peso total y estimar el centro de masa de robots competitivos mediante celdas de carga, módulos HX711 y Arduino Nano.

10.5.2. Objetivos específicos

1. Conectar celdas de carga a módulos HX711.
2. Calibrar cada celda utilizando una masa conocida.

3. Medir la distribución de peso en cuatro puntos de apoyo.
4. Calcular el centro de masa aproximado del robot.
5. Analizar cómo cambia el centro de masa al mover baterías, motores o accesorios.
6. Registrar resultados experimentales para comparar configuraciones mecánicas.

10.6. Materiales y herramientas

Tabla 10.1: Materiales principales para la plataforma de medición.

Elemento	Cantidad	Función
Arduino Nano	1	Procesamiento de mediciones
Celdas de carga	4	Medición de peso por esquina
Módulos HX711	4	Amplificación y conversión ADC
Base rígida	1	Soporte de la plataforma
Placa superior	1	Superficie donde se coloca el robot
Cables Dupont o soldados	Varios	Interconexión
Masas conocidas	Varias	Calibración
Computadora con Arduino IDE	1	Programación y registro serial
Robot de prueba	1	Objeto a medir

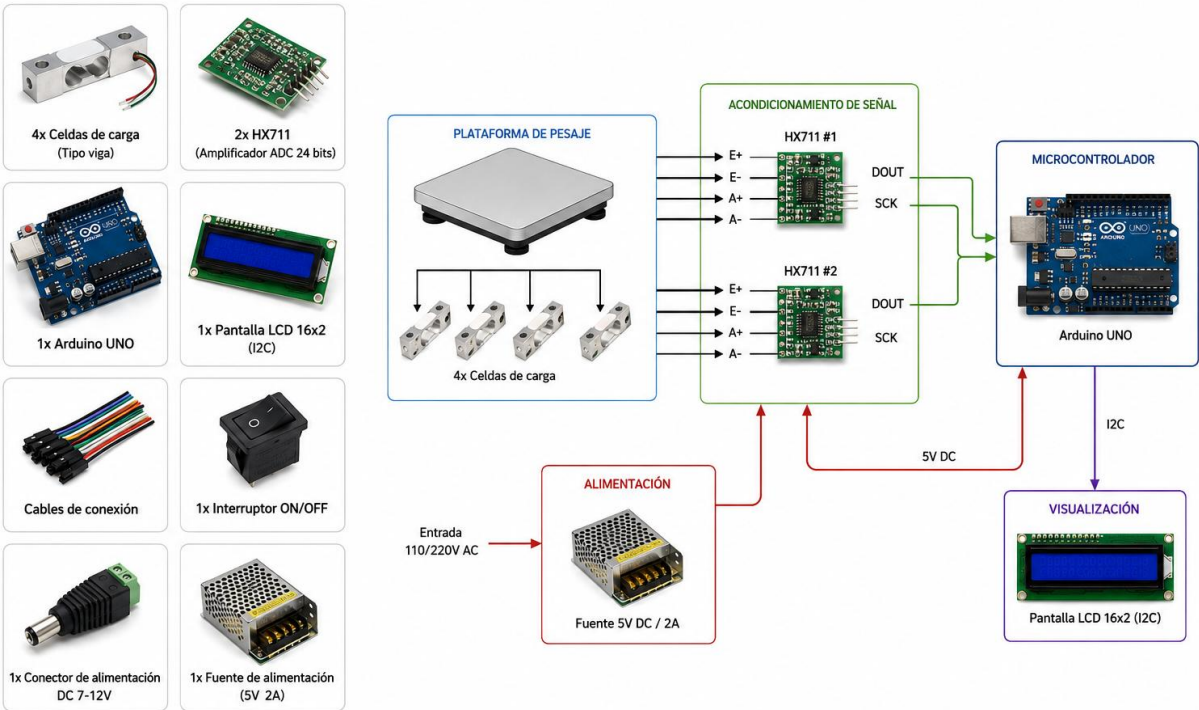


Figura 10.3: Materiales utilizados para construir la plataforma de medición de centro de masa.

10.7. Fundamento teórico

10.7.1. Centro de masa

El centro de masa es el punto donde puede considerarse concentrada la masa total de un cuerpo para efectos de análisis de traslación. En un robot apoyado sobre una superficie horizontal, la posición del centro de masa proyectada sobre el plano puede estimarse a partir de la distribución de carga en sus puntos de apoyo [7].

Si un robot está colocado sobre cuatro sensores ubicados en una plataforma rectangular, cada sensor mide una fuerza proporcional al peso soportado en esa esquina. Con estas fuerzas se puede estimar la posición del centro de masa.

10.7.2. Sumatoria de fuerzas

Si las cuatro celdas de carga miden fuerzas F_1 , F_2 , F_3 y F_4 , el peso total se aproxima mediante:

$$F_T = F_1 + F_2 + F_3 + F_4$$

donde F_T es la fuerza total equivalente al peso del robot sobre la plataforma.

10.7.3. Cálculo de coordenadas del centro de masa

Supóngase una plataforma rectangular de largo L y ancho W . Se define el origen en la esquina posterior izquierda. Las coordenadas de las celdas son:

- Posterior izquierda: (0, 0)
- Posterior derecha: (W, 0)
- Frontal izquierda: (0, L)
- Frontal derecha: (W, L)

La coordenada x del centro de masa se calcula como:

$$x_{cm} = \frac{F_{PD}W + F_{FD}W}{F_T}$$

La coordenada y se calcula como:

$$y_{cm} = \frac{F_{FI}L + F_{FD}L}{F_T}$$

donde:

- F_{PI} : fuerza posterior izquierda.
- F_{PD} : fuerza posterior derecha.
- F_{FI} : fuerza frontal izquierda.
- F_{FD} : fuerza frontal derecha.

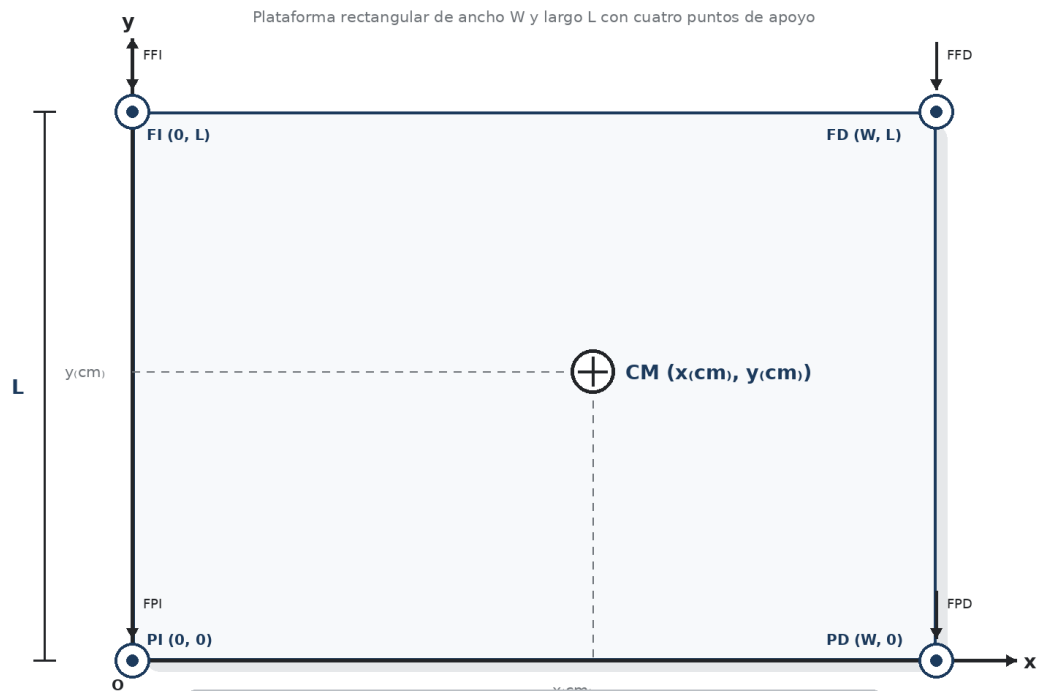


Figura 10.4: Modelo geométrico para estimar el centro de masa sobre una plataforma

10.7.4. Celda de carga

Una celda de carga es un transductor que convierte una fuerza mecánica en una señal eléctrica. Muchas celdas de carga emplean galgas extensométricas conectadas en configuración de puente de Wheatstone. Cuando se aplica una fuerza, la deformación mecánica produce un cambio pequeño de resistencia, generando una señal diferencial de muy bajo nivel.

10.7.5. Módulo HX711

El HX711 es un convertidor analógico-digital de 24 bits diseñado para aplicaciones de pesaje. Incluye un amplificador de ganancia programable y se utiliza frecuentemente con celdas de carga [1]. SparkFun y otros fabricantes ofrecen guías de conexión para utilizar celdas de carga con módulos basados en HX711 [2].

El módulo utiliza dos líneas principales hacia el microcontrolador:

- **DT o DOUT:** salida de datos.
- **SCK:** reloj de lectura.

10.8. Diseño electrónico

Cada celda de carga se conecta a un módulo HX711. Para evitar interferencias y lecturas inestables, se recomienda mantener cables cortos, conexiones firmes y una base mecánica rígida.

Tabla 10.2: Asignación sugerida de pines para cuatro módulos HX711.

Módulo	DOUT	SCK
Posterior izquierdo	D2	D3
Posterior derecho	D4	D5
Frontal izquierdo	D6	D7
Frontal derecho	D8	D9

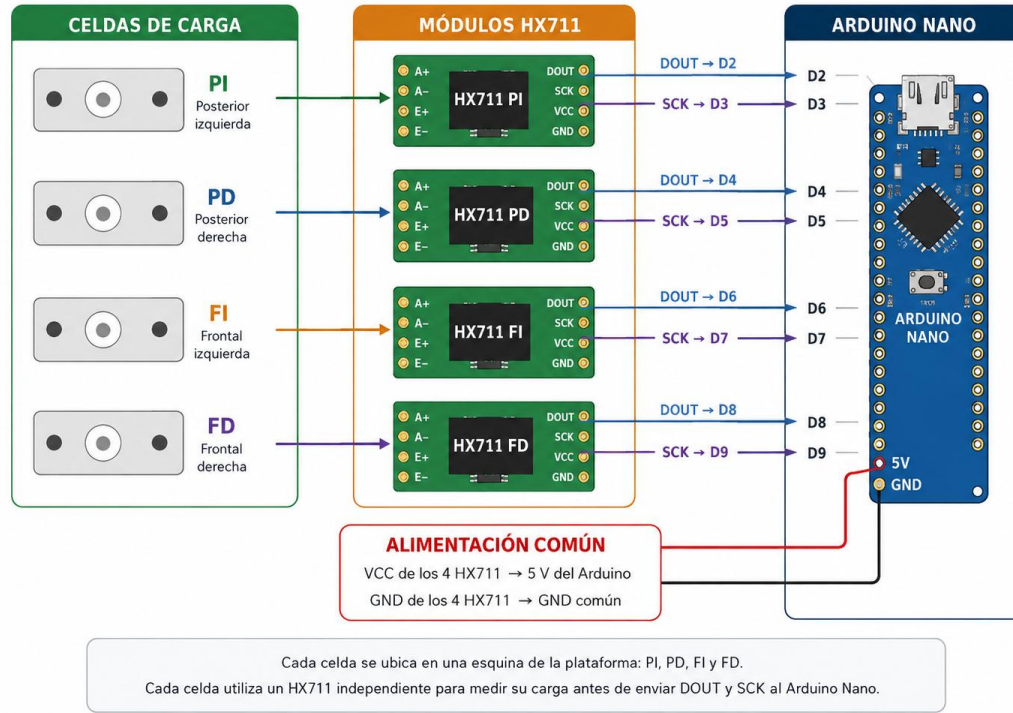


Figura 10.5: Diagrama de conexiones de la plataforma de medición.



Figura 10.6: Montaje electrónico de la plataforma de medición de centro de masa.

10.9. Diseño mecánico de la plataforma

La precisión del sistema depende en gran medida del montaje mecánico. Las celdas deben estar correctamente apoyadas y la plataforma superior debe transmitir la carga de forma estable.

Recomendaciones:

- Utilizar una base inferior rígida.
- Colocar cada celda en una esquina de la plataforma.
- Asegurar que la placa superior apoye correctamente sobre las celdas.
- Evitar que la plataforma roce con elementos externos.
- Nivelar la superficie antes de calibrar.
- Marcar el sistema de coordenadas x e y.

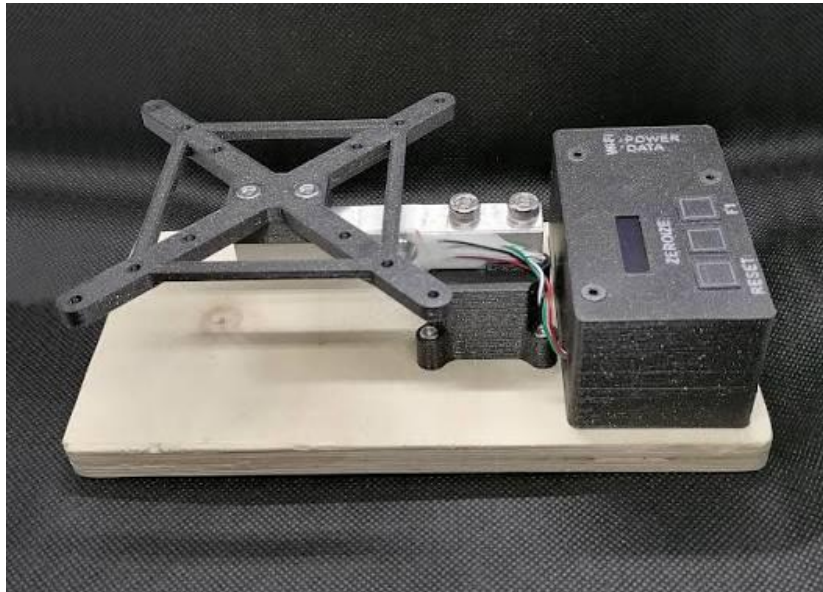


Figura 10.7: Diseño mecánico sugerido para la plataforma de centro de masa.

10.10. Calibración de celdas de carga

Cada celda debe calibrarse individualmente. Para ello se utiliza una masa conocida, se coloca sobre la celda correspondiente y se ajusta el factor de calibración hasta que la lectura coincida con el valor real.

Se recomienda realizar la calibración en etapas:

1. Encender el sistema sin carga.
2. Ejecutar la función de tara.
3. Colocar una masa conocida.
4. Ajustar el factor de calibración.
5. Repetir el proceso en cada celda.

Tabla 10.3: Formato sugerido para calibración de celdas.

Celda	Masa conocida	Lectura obtenida	Factor calibración
Posterior izquierda			
Posterior derecha			
Frontal izquierda			
Frontal derecha			

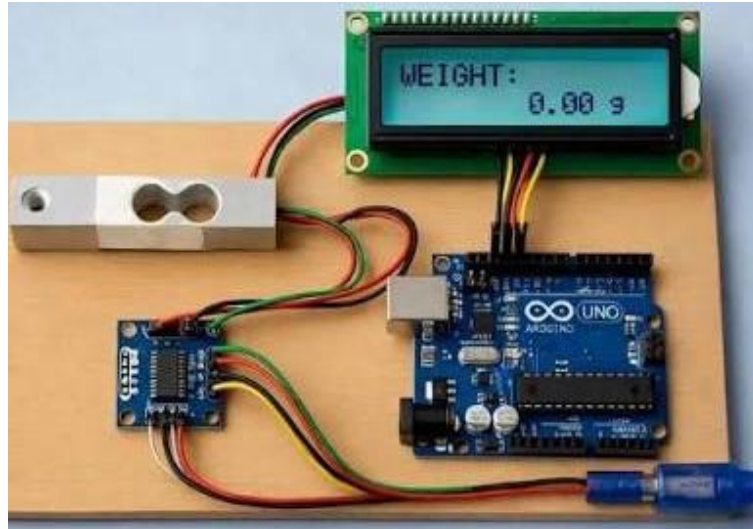


Figura 10.8: Calibración individual de celdas de carga.

10.11. Código de calibración

El siguiente programa permite leer una celda de carga mediante un HX711. Debe ejecutarse individualmente para cada celda y utilizarse para encontrar el factor de calibración [3].

```
#include "HX711.h"

const byte PIN_DOUT = 2;
const byte PIN_SCK = 3;

HX711 balanza;

float factorCalibracion = -7050.0; // ajustar experimentalmente

void setup()
{
  Serial.begin(115200);

  balanza.begin(PIN_DOUT, PIN_SCK);
  balanza.set_scale(factorCalibracion);
```

```

balanza.tare();

Serial.println("Calibracion de celda iniciada");
Serial.println("Coloque una masa conocida y ajuste el factor.");
}
void loop()
{
    float masa = balanza.get_units(10);

    Serial.print("Lectura: ");
    Serial.print(masa, 2);
    Serial.println(" unidades");

    delay(500);
}

```

10.12. Código del proyecto

El siguiente programa utiliza cuatro módulos HX711 para estimar el peso total y el centro de masa en una plataforma rectangular.

```

#include "HX711.h"

// =====
// DIMENSIONES DE LA PLATAFORMA
// =====
const float ANCHO_W = 20.0; // cm
const float LARGO_L = 25.0; // cm

// =====
// MODULOS HX711
// =====
HX711 celdaPI; // posterior izquierda
HX711 celdaPD; // posterior derecha
HX711 celdaFI; // frontal izquierda
HX711 celdaFD; // frontal derecha

// Pines: DOUT, SCK
const byte PI_DOUT = 2;
const byte PI_SCK = 3;

const byte PD_DOUT = 4;
const byte PD_SCK = 5;
const byte FI_DOUT = 6;
const byte FI_SCK = 7;

const byte FD_DOUT = 8;

```

```

const byte FD_SCK = 9;

// Factores de calibracion
float factorPI = -7050.0;
float factorPD = -7050.0;
float factorFI = -7050.0;
float factorFD = -7050.0;

// =====
// VARIABLES
// =====

float F_PI, F_PD, F_FI, F_FD;
float F_total;
float x_cm, y_cm;

// =====
// FUNCIONES
// =====

void iniciarCeldas()
{
    celdaPI.begin(PI_DOUT, PI_SCK);
    celdaPD.begin(PD_DOUT, PD_SCK);
    celdaFI.begin(FI_DOUT, FI_SCK);
    celdaFD.begin(FD_DOUT, FD_SCK);

    celdaPI.set_scale(factorPI);
    celdaPD.set_scale(factorPD);
    celdaFI.set_scale(factorFI);
    celdaFD.set_scale(factorFD);
}

void hacerTara()
{
    celdaPI.tare();
    celdaPD.tare();
    celdaFI.tare();
    celdaFD.tare();
}

void leerFuerzas()
{
    F_PI = celdaPI.get_units(5);
    F_PD = celdaPD.get_units(5);
    F_FI = celdaFI.get_units(5);
    F_FD = celdaFD.get_units(5);

    // Evitar valores negativos pequeños por ruido

```

```

    if (F_PI < 0) F_PI = 0;
    if (F_PD < 0) F_PD = 0;
    if (F_FI < 0) F_FI = 0;
    if (F_FD < 0) F_FD = 0;
}

void calcularCentroMasa()
{
    F_total = F_PI + F_PD + F_FI + F_FD;

    if (F_total > 1.0)
    {
        x_cm = ((F_PD * ANCHO_W) + (F_FD * ANCHO_W)) / F_total;
        y_cm = ((F_FI * LARGO_L) + (F_FD * LARGO_L)) / F_total;
    }
    else
    {
        x_cm = 0;
        y_cm = 0;
    }
}

void enviarDatos()
{
    Serial.print("PI:");
    Serial.print(F_PI, 2);

    Serial.print(" PD:");
    Serial.print(F_PD, 2);

    Serial.print(" FI:");
    Serial.print(F_FI, 2);

    Serial.print(" FD:");
    Serial.print(F_FD, 2);

    Serial.print(" TOTAL:");
    Serial.print(F_total, 2);

    Serial.print(" Xcm:");
    Serial.print(x_cm, 2);

    Serial.print(" Ycm:");
    Serial.println(y_cm, 2);
}

// =====

```

```
// SETUP Y LOOP
// =====
void setup()
{
  Serial.begin(115200);

  iniciarCeldas();

  Serial.println("Retire toda carga de la plataforma...");
  delay(3000);

  hacerTara();

  Serial.println("Sistema de medicion de centro de masa iniciado");
}

void loop()
{
  leerFuerzas();
  calcularCentroMasa();
  enviarDatos();

  delay(300);
}
```

Los factores de calibración incluidos son referenciales. Cada celda y cada módulo HX711 deben calibrarse con masas conocidas antes de realizar mediciones confiables.

10.13. Interpretación de resultados

El valor x_{cm} indica qué tan desplazado está el centro de masa hacia la derecha o izquierda de la plataforma. El valor y_{cm} indica si el centro de masa está más cerca de la parte frontal o posterior.

Tabla 10.4: Interpretación básica de la posición del centro de masa.

Resultado	Interpretación
x_{cm} cercano a $W/2$	Distribución lateral equilibrada
x_{cm} menor a $W/2$	Mayor carga hacia la izquierda
x_{cm} mayor a $W/2$	Mayor carga hacia la derecha
y_{cm} cercano a $W/2$	Distribución longitudinal centrada
y_{cm} mayor a $W/2$	Mayor carga hacia el frente
y_{cm} menor a $W/2$	Mayor carga hacia la parte posterior

10.14. Aplicación en Minisumo

En Minisumo, la distribución de masa debe favorecer la tracción y el ataque. Si el robot utiliza tracción trasera, puede convenir que una parte importante del peso esté sobre las ruedas motrices. Si el diseño depende de una cuña frontal, también es importante que el frente mantenga contacto adecuado con el dohyo.

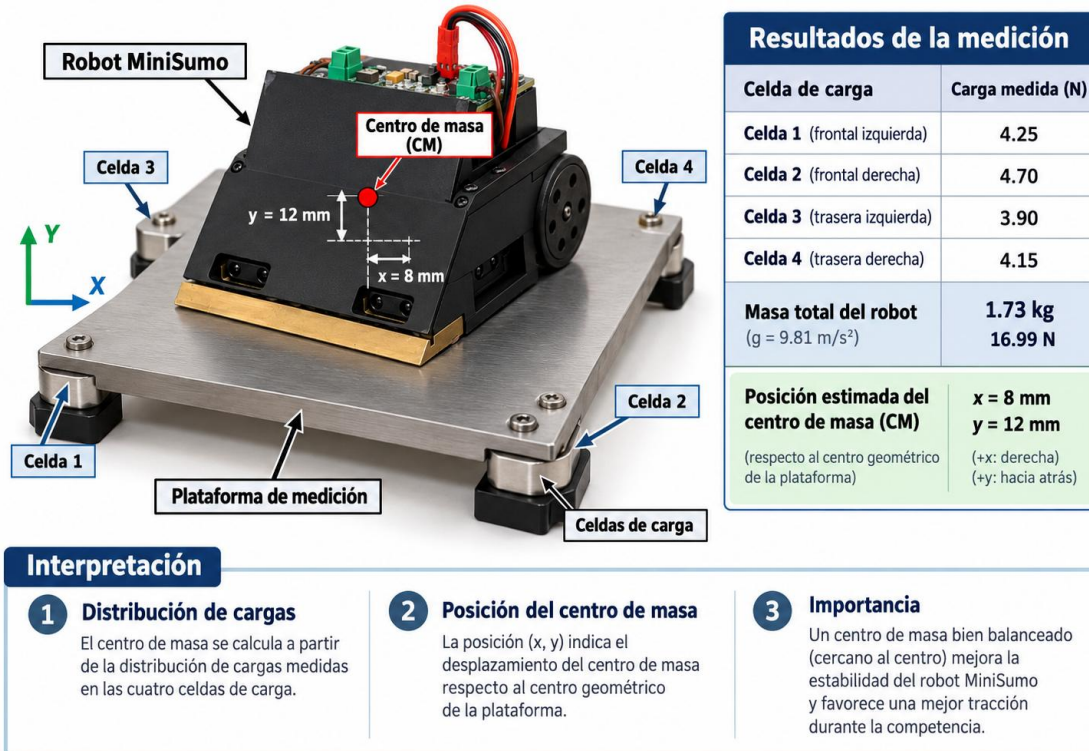


Figura 10.9: Evaluación del centro de masa en un robot Minisumo.

Experimentos sugeridos:

1. Medir el robot con la batería en posición central.
2. Mover la batería hacia el frente y repetir la medición.
3. Mover la batería hacia atrás y repetir la medición.
4. Comparar tracción, capacidad de empuje y estabilidad.

10.15. Aplicación en seguidores de línea

En un seguidor de línea, el centro de masa influye en la respuesta dinámica. Un centro de masa demasiado alto puede aumentar la inestabilidad en curvas. Un centro de masa desplazado hacia un lado puede generar diferencias en la adherencia de las ruedas [6].

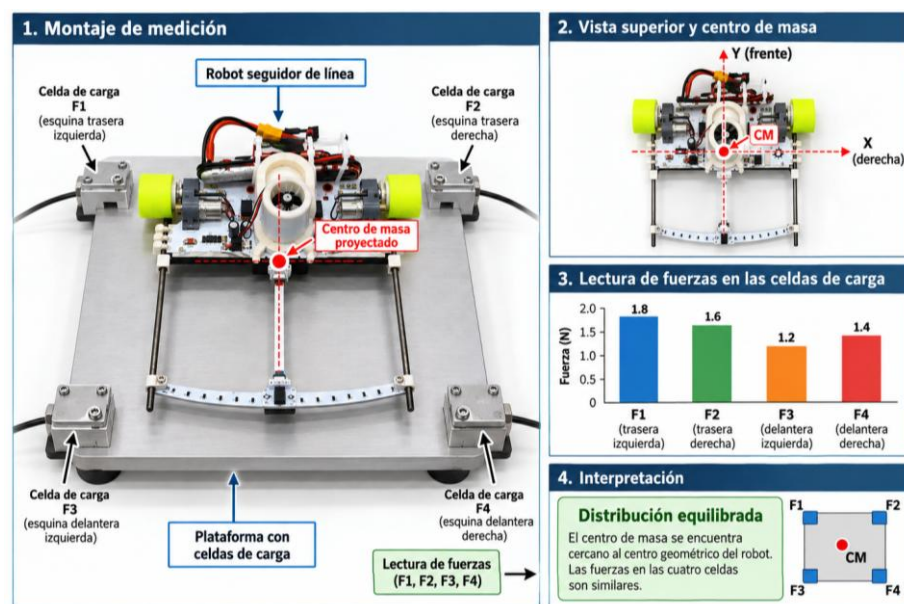


Figura 10.10: Evaluación de centro de masa en un robot seguidor de línea.

10.16. Pruebas experimentales

10.16.1. Prueba 1: peso total

Colocar una masa conocida sobre la plataforma y verificar que la suma de las cuatro celdas sea cercana al valor real.

10.16.2. Prueba 2: desplazamiento lateral

Colocar el mismo objeto hacia la izquierda y luego hacia la derecha. Verificar que x_{cm} cambie de forma coherente.

10.16.3. Prueba 3: desplazamiento longitudinal

Colocar el objeto hacia el frente y luego hacia atrás. Verificar que y_{cm} cambie de forma coherente.

10.16.4. Prueba 4: robot real

Medir el robot en diferentes configuraciones de batería, sensores o accesorios.

Configuración	Peso total	x_{cm}	y_{cm}	Observación
Batería central				
Batería frontal				
Batería posterior				
Peso posterior				
Peso adicional frontal				
Peso adicional lateral				

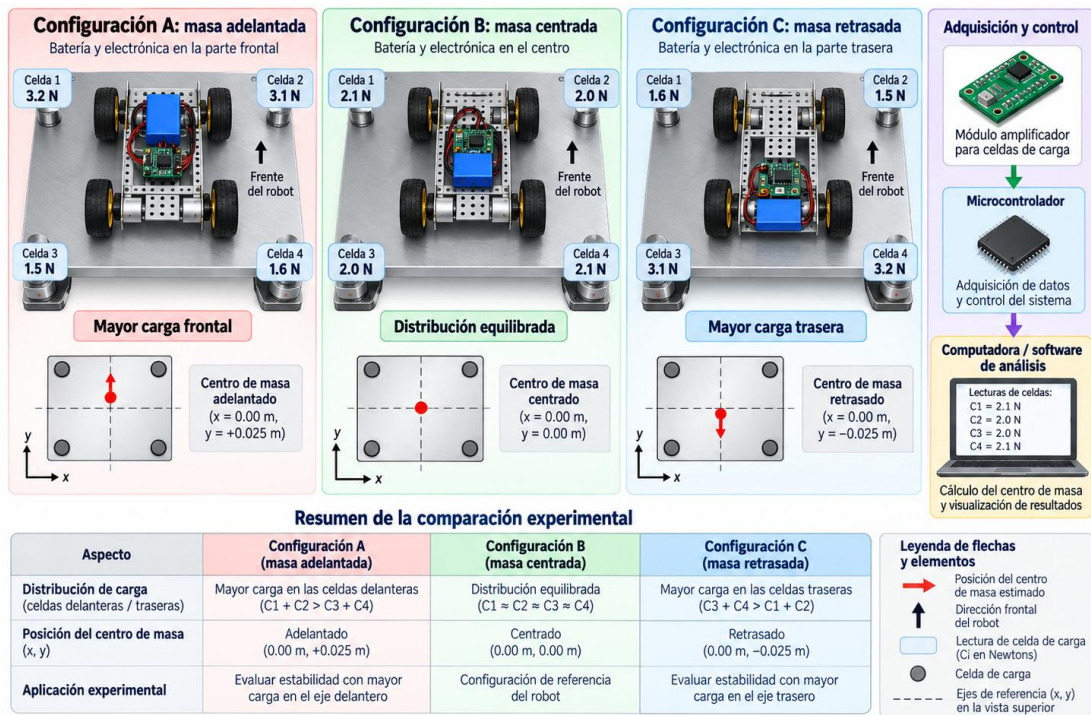


Figura 10.11: Comparación experimental de configuraciones mecánicas.

10.17. Análisis de datos

Los datos del monitor serial pueden copiarse a una hoja de cálculo para generar gráficos. Una representación útil consiste en dibujar la plataforma como un rectángulo y marcar el punto (x_{cm} , y_{cm}) de cada configuración.

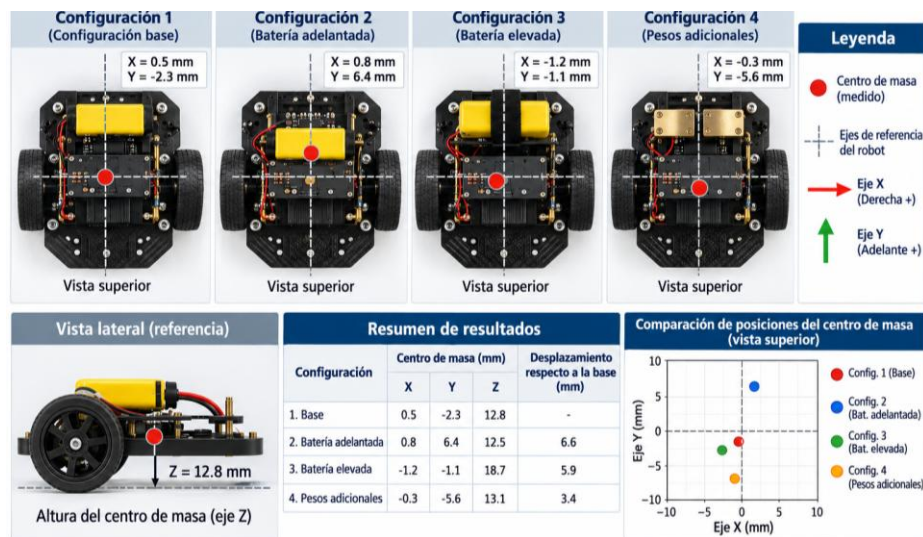


Figura 10.12: Visualización comparativa del centro de masa en diferentes configuraciones.

10.18. Errores frecuentes

Durante el desarrollo pueden presentarse los siguientes errores:

- Celdas de carga mal orientadas mecánicamente.
- Plataforma superior rozando con la base inferior.
- No realizar tara antes de medir.
- Usar factores de calibración iguales sin calibrar cada celda.
- Cables flojos o mal soldados en el puente de la celda.
- Lecturas negativas por ruido o mala calibración.
- Colocar el robot fuera del área útil de la plataforma.
- Confundir las posiciones frontal, posterior, izquierda y derecha.

10.19. Proyecto de extensión

Como extensión, se propone desarrollar una versión con ESP32 y visualización web. El sistema podría mostrar en tiempo real un plano de la plataforma y un punto móvil indicando el centro de masa. También podría guardar configuraciones, generar reportes y comparar robots diferentes.

Otra extensión consiste en combinar esta plataforma con el sistema de telemetría de impactos del Capítulo 5 para relacionar distribución de masa con estabilidad ante golpes o vibraciones.



Figura 10.13: Extensión con ESP32 y visualización web del centro de masa.

Actividades de aprendizaje

1. Calibre una celda de carga utilizando una masa conocida.
2. Construya una plataforma con cuatro celdas de carga.
3. Mida el centro de masa de un robot con batería en tres posiciones diferentes.
4. Genere una gráfica con los puntos de centro de masa obtenidos.
5. Analice cómo cambia la tracción del robot al modificar la distribución de peso.
6. Proponga una versión del sistema con ESP32 y visualización web.

Preguntas de reflexión

1. ¿Por qué el centro de masa influye en la tracción de un Minisumo?
2. ¿Qué problemas puede generar una distribución lateral desbalanceada?
3. ¿Por qué es necesario calibrar cada celda de carga de forma individual?
4. ¿Cómo afecta la altura del centro de masa a la estabilidad de un robot?
5. ¿Qué configuración de centro de masa sería conveniente para un seguidor de línea rápido?

Referencias

- [1] Avia Semiconductor, “HX711 24-Bit Analog-to-Digital Converter for Weigh Scales”, Datasheet. [En línea]. Disponible en: https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf. [Accedido: 24-jun-2026].
- [2] SparkFun, “Load Cell Amplifier HX711 Breakout Hookup Guide#”, SparkFun Learn. [En línea]. Disponible en: <https://learn.sparkfun.com/tutorials/load-cell-amplifier-hx711-breakout-hookup-guide/all>. [Accedido: 24-jun2026].
- [3] B. Cort, “HX711 Arduino Library”, GitHub. [En línea]. Disponible en: <https://github.com/bogde/HX711>. [Accedido: 24-jun-2026].
- [4] Arduino, “Nano”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/hardware/nano>. [Accedido: 24-jun-2026].
- [5] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed. Cham, Switzerland: Springer, 2016.
- [6] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.
- [7] R. C. Hibbeler, *Engineering Mechanics: Statics*, 14th ed. Hoboken, NJ, USA: Pearson, 2016.

Capítulo 11. Seguidor de Línea Competitivo con PID y Barra de 16 Sensores

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender la arquitectura general de un robot seguidor de línea competitivo.
- Leer una barra de sensores reflectivos multiplexada de 16 canales.
- Calibrar valores mínimos y máximos de cada sensor sobre pista blanca y línea negra.
- Calcular la posición relativa de la línea mediante promedio ponderado.
- Implementar un controlador PID para corregir la trayectoria.
- Ajustar velocidad base, freno, ganancia proporcional, derivativa e integral mediante pruebas experimentales.
- Identificar problemas comunes como oscilación, apertura en curvas, pérdida de línea y sensibilidad a la iluminación.

11.1. Introducción

El robot seguidor de línea es una de las plataformas más utilizadas para introducir conceptos de sensores, control, programación y diseño mecánico. En su versión básica, el robot detecta una línea sobre una pista y ajusta la velocidad de sus motores para mantenerse sobre la trayectoria. Sin embargo, en competencias de alto nivel, esta tarea se vuelve considerablemente más compleja [6].

Un seguidor competitivo debe desplazarse a alta velocidad, reaccionar ante curvas cerradas, interpretar cruces, resistir variaciones de iluminación, mantener estabilidad mecánica y recuperar la línea cuando existe una pérdida temporal. La diferencia entre un robot educativo y un robot competitivo no está únicamente en la velocidad de los motores, sino en la calidad de la lectura de sensores, el ajuste del control, la rigidez del chasis, la adherencia de las ruedas y la estrategia de frenado.

En este capítulo se desarrolla un robot seguidor de línea competitivo utilizando una barra de 16 sensores reflectivos y control PID. El proyecto se basa en una arquitectura común en robots de competencia: una barra frontal de sensores, dos motores DC con caja reductora, un driver de motores, una batería LiPo y un microcontrolador Arduino compatible.

El objetivo es construir una base de programación que permita leer sensores, calibrarlos, calcular el error respecto al centro de la línea y aplicar una corrección diferencial sobre los motores. Además, se incorporan criterios de ajuste experimental útiles para situaciones reales de competencia, como curvas cerradas, cambios de iluminación y pérdida momentánea de línea.

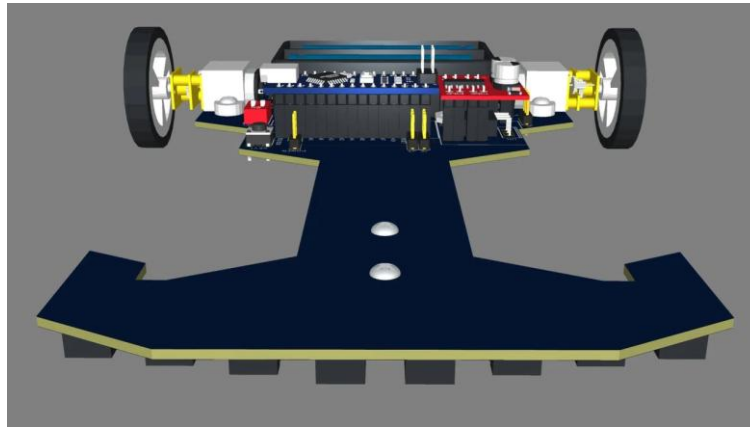


Figura 11.1: Robot seguidor de línea competitivo con barra frontal de sensores.

11.2. Planteamiento del problema

Muchos seguidores de línea básicos utilizan dos o tres sensores digitales y reglas simples como: si detecta a la izquierda, gira a la izquierda; si detecta a la derecha, gira a la derecha; si detecta al centro, avanza. Esta estrategia funciona a baja velocidad, pero presenta limitaciones importantes cuando la pista tiene curvas cerradas, cambios bruscos, iluminación irregular o tramos rápidos.

Los problemas más frecuentes en seguidores competitivos son:

- Oscilación excesiva en rectas.
- Apertura en curvas después de una recta rápida.
- Salida de pista en curvas cerradas.
- Reacción tardía por sensores mal calibrados.
- Pérdida de línea por vibración de la barra.
- Diferencia de velocidad entre motores.
- Frenado insuficiente al entrar en curvas de 90 grados.
- Lecturas falsas por iluminación o altura variable.

Para resolver estos problemas se requiere una lectura de línea más precisa y un controlador que no solo indique hacia dónde girar, sino cuánto corregir.

Un seguidor competitivo no debe decidir únicamente con sensores digitales. Debe estimar la posición de la línea de forma continua y aplicar una corrección proporcional al error.

11.3. Descripción general del proyecto

El robot propuesto utiliza una barra de 16 sensores reflectivos conectada al Arduino mediante multiplexación. Cada sensor entrega una lectura analógica asociada a la reflexión de la

superficie. Con estas lecturas se calcula la posición de la línea y se obtiene un error respecto al centro de la barra.

El controlador PID utiliza ese error para modificar la velocidad de los motores:

$$PWM_{izq} = V_{base} + correccion$$

$$PWM_{der} = V_{base} - correccion$$

Si la línea se desplaza hacia un lado, el robot reduce o invierte parcialmente un motor y aumenta el otro, generando el giro necesario para recuperar la trayectoria [3, 4].

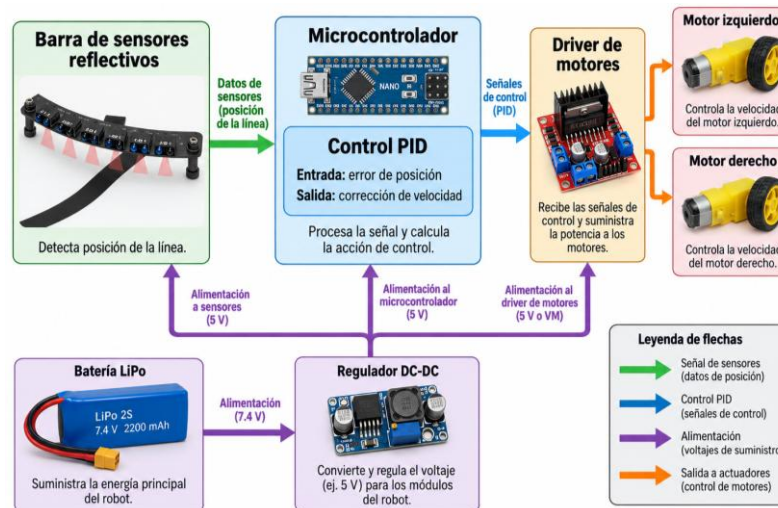


Figura 11.2: Arquitectura general del seguidor de línea competitivo con PID.

11.4. Aplicación STEM

Este proyecto integra varias dimensiones STEM:

- **Ciencia:** reflexión de luz, fricción, aceleración, adherencia y dinámica del movimiento.
- **Tecnología:** sensores reflectivos, multiplexación, microcontroladores, drivers de motor y baterías LiPo.
- **Ingeniería:** diseño de un sistema móvil rápido, estable y ajustable para competencia.
- **Matemáticas:** promedio ponderado, error, derivada, integral, saturación y ajuste de parámetros.

11.5. Objetivos del proyecto

11.5.1. Objetivo general

Diseñar e implementar un robot seguidor de línea competitivo basado en barra de 16 sensores y control PID, capaz de seguir una línea a velocidad ajustable con respuesta estable en rectas y curvas.

11.5.2. Objetivos específicos

- 1. Leer una barra de 16 sensores reflectivos mediante selección multiplexada.
- 2. Calibrar valores mínimos y máximos de cada sensor.
- 3. Normalizar lecturas para reducir diferencias entre sensores.
- 4. Calcular la posición de la línea mediante promedio ponderado.
- 5. Implementar control PID sobre motores diferenciales.
- 6. Ajustar parámetros para mejorar respuesta en rectas, curvas y cambios de iluminación.

11.6. Materiales y herramientas

Tabla 11.1: Materiales principales para el seguidor de línea competitivo.

Elemento	Cantidad	Función
Arduino Nano o compatible	1	Control principal
Barra de 16 sensores reflectivos	1	Detección de línea
Driver de motores TB6612FNG o similar	1	Control de dos motores DC
Motores DC con caja reductora	2	Tracción diferencial
Ruedas de alta adherencia	2	Contacto con la pista
Batería LiPo 2S	1	Alimentación del robot
Regulador DC-DC	1	Alimentación estable para control
Chasis liviano y rígido	1	Estructura mecánica
Interruptor general	1	Encendido y seguridad
Pista de línea	1	Pruebas experimentales

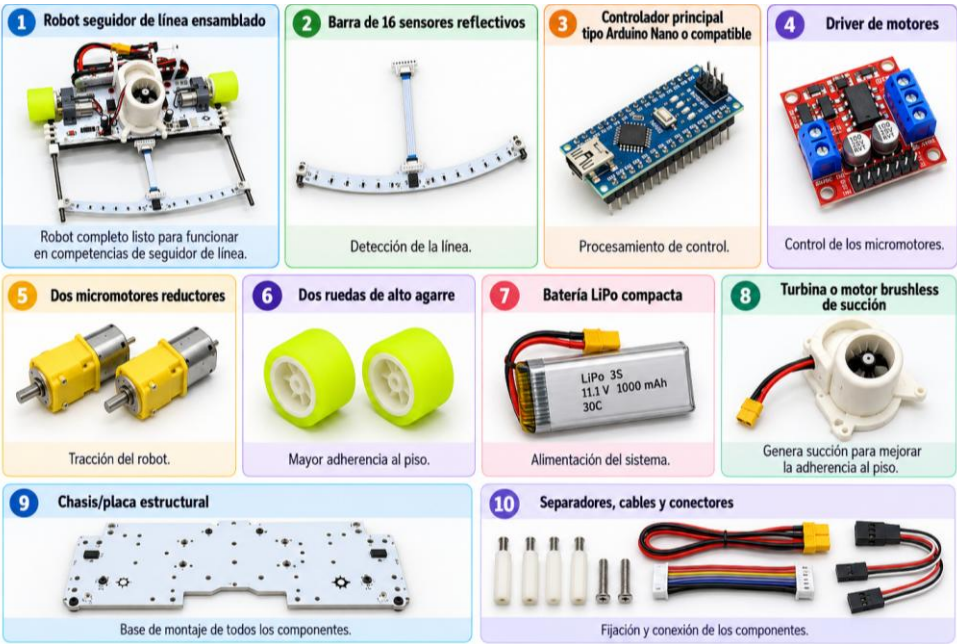


Figura 11.3: Materiales utilizados para construir el seguidor de línea competitivo.

11.7. Fundamento teórico

11.7.1. Sensores reflectivos para seguimiento de línea

Los sensores reflectivos detectan la diferencia de reflexión entre la línea y el fondo de la pista. Una superficie clara refleja más luz infrarroja que una superficie oscura. En seguidores de línea, esta diferencia se convierte en una señal analógica o digital que el microcontrolador puede interpretar.

Los arreglos de sensores de reflectancia, como los sensores QTR, son utilizados ampliamente en robots seguidores de línea y aplicaciones de detección de superficies [5]. El principio también se aplica a barras de sensores multiplexadas utilizadas en robots de competencia [1].

11.7.2. Multiplexación de sensores

Una barra de 16 sensores puede entregar una sola salida analógica y cuatro líneas de selección. Las líneas S0, S1, S2 y S3 seleccionan cuál de los 16 sensores se conecta a la salida analógica.

Para seleccionar el sensor i , se escribe el valor binario de i en las líneas de selección.

Por ejemplo:

$$i = 5 \Rightarrow 0101_2$$

Esto significa que $S_0 = 1$, $S_1 = 0$, $S_2 = 1$, $S_3 = 0$, dependiendo del orden físico de conexión.

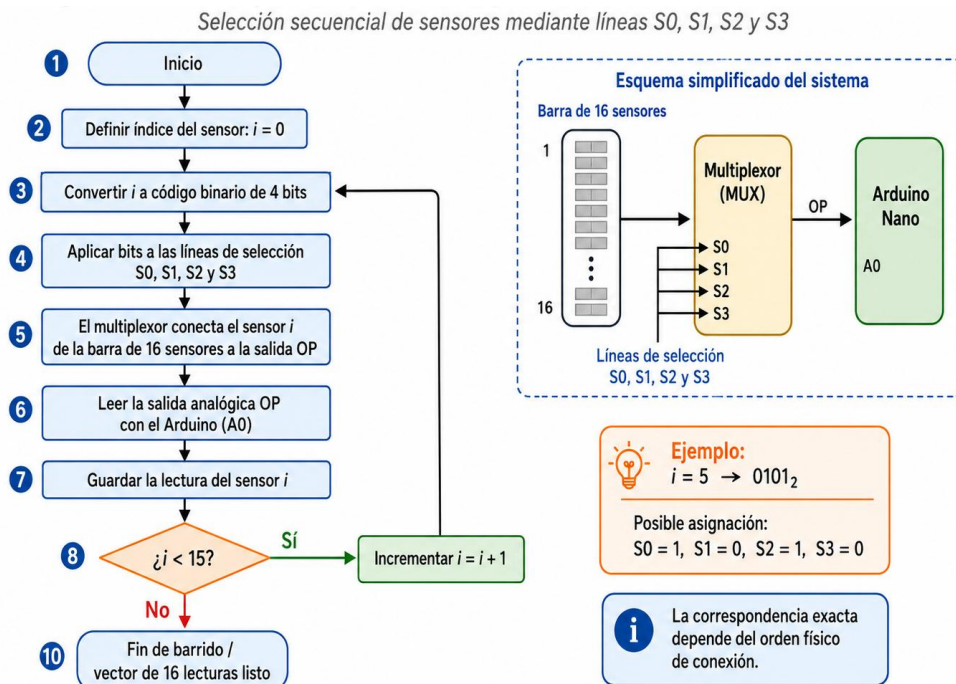


Figura 11.4: Principio de lectura multiplexada de una barra de 16 sensores.

11.7.3. Normalización de lecturas

Cada sensor puede entregar valores distintos debido a tolerancias, altura, suciedad o diferencias de fabricación. Para comparar sensores de forma justa, se recomienda calibrar mínimo y máximo [2]:

$$valor_{norm} = \frac{valor - min}{max - min}$$

En Arduino se puede escalar este resultado a un rango de 0 a 1000:

$$valor_{norm} = 1000 \cdot \frac{valor - min}{max - min}$$

11.7.4. Cálculo de posición de línea

Una forma común de estimar la posición de la línea es utilizar promedio ponderado.

Cada sensor tiene una posición asignada. Si hay 16 sensores, se pueden asignar posiciones de 0 a 15000:

$$posicion = \frac{\sum_{i=0}^{15} lectura_i \cdot posicion_i}{\sum_{i=0}^{15} lectura_i}$$

El centro ideal de la barra será:

$$centro = 7500$$

El error se calcula como:

$$error = posicion - centro$$

Si el error es negativo, la línea está hacia un lado; si es positivo, está hacia el lado contrario. La convención exacta depende del orden físico de los sensores.

11.7.5. Control PID

El controlador PID combina tres términos:

$$u(t) = K_p e(t) + K_i \int e(t) dt + K_d \frac{de(t)}{dt}$$

En un microcontrolador se implementa de forma discreta:

$$P = K_p e$$

$$I = I + e$$

$$D = e - e_{anterior}$$

$$correccion = P + K_i I + K_d D$$

El término proporcional reacciona al error actual. El término derivativo responde al cambio del error y ayuda a reducir oscilaciones. El término integral corrige errores acumulados, aunque en seguidores de línea rápidos suele usarse con valores muy pequeños o incluso desactivarse [5].

11.8. Diseño electrónico

Tabla 11.2: Asignación sugerida de pines para barra multiplexada y motores.

Pin Arduino	Conexión	Función
A0	S0 barra sensores	Selección bit 0
A1	S1 barra sensores	Selección bit 1
A2	S2 barra sensores	Selección bit 2
A3	S3 barra sensores	Selección bit 3
A7	OP barra sensores	Lectura analógica
D3	PWMA driver	PWM motor izquierdo
D4	AIN1 driver	Dirección motor izquierdo
D5	AIN2 driver	Dirección motor izquierdo
D6	PWMB driver	PWM motor derecho
D7	BIN1 driver	Dirección motor derecho
D8	BIN2 driver	Dirección motor derecho
D9	STBY driver	Habilitación TB6612FNG

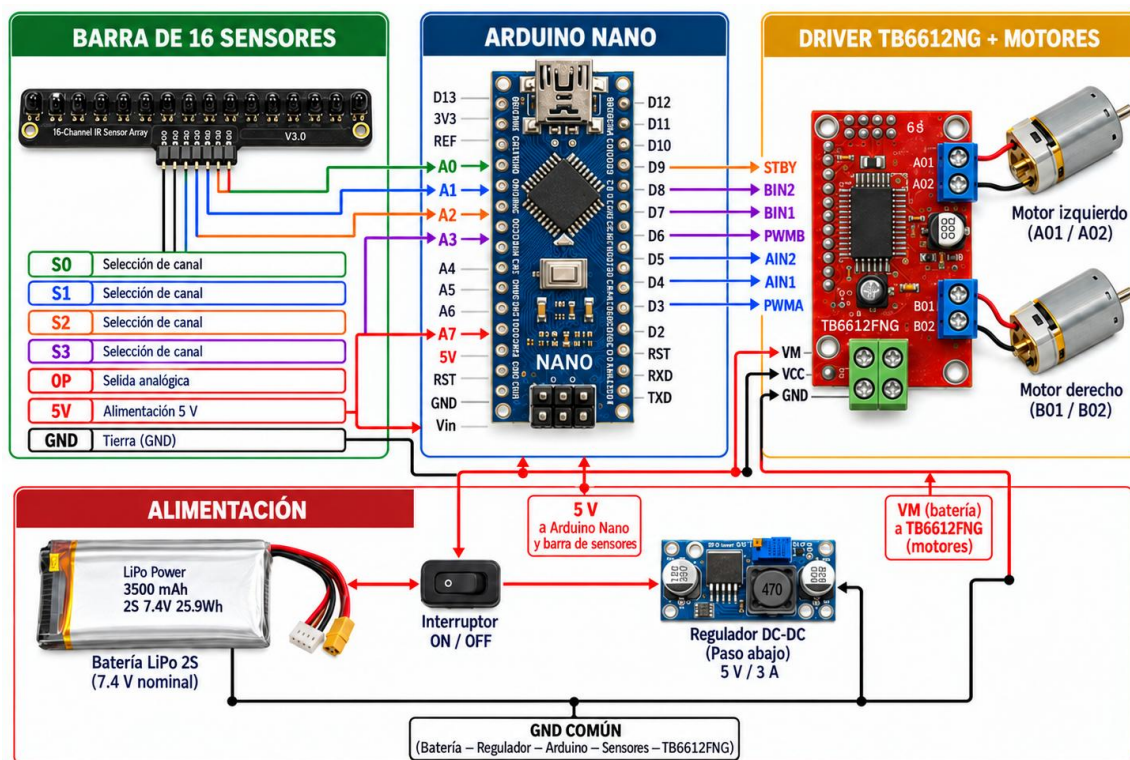


Figura 11.5: Diagrama de conexiones del seguidor de línea con barra de 16 sensores.

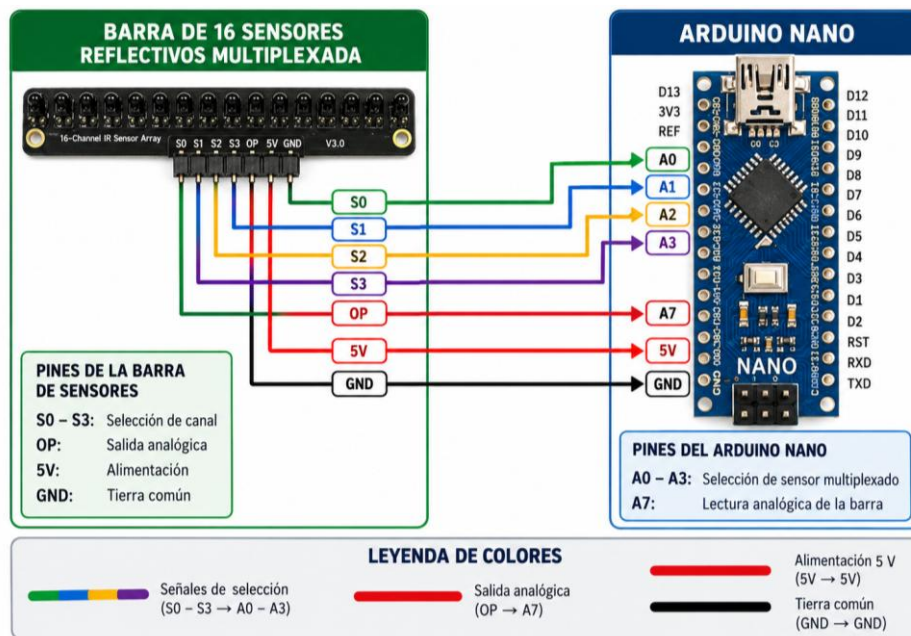


Figura 11.6: Cableado de la barra de sensores reflectivos multiplexada.

11.9. Calibración de la barra de sensores

La calibración debe realizarse sobre la pista real. El robot debe moverse de forma que todos los sensores pasen sobre línea negra y fondo blanco. Durante este proceso se registran los valores mínimos y máximos de cada canal.

```
const byte S0 = A0;
const byte S1 = A1;
const byte S2 = A2;
const byte S3 = A3;
const byte OP = A7;

void seleccionarSensor(byte canal)
{
  digitalWrite(S0, bitRead(canal, 0));
  digitalWrite(S1, bitRead(canal, 1));
  digitalWrite(S2, bitRead(canal, 2));
  digitalWrite(S3, bitRead(canal, 3));
  delayMicroseconds(50);
}

void setup()
{
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
```



```

pinMode(S3, OUTPUT);

Serial.begin(115200);
}

void loop()
{
  for (byte i = 0; i < 16; i++)
  {
    seleccionarSensor(i);
    int valor = analogRead(OP);

    Serial.print(valor);
    Serial.print('\t');
  }

  Serial.println();
  delay(100);
}

```

Tabla 11.3: Formato sugerido para calibración de la barra de sensores.

Sensor	Mínimo	Máximo	Observación
S0			
S1			
S2			
S3			
S4 – S15			Completar para cada sensor

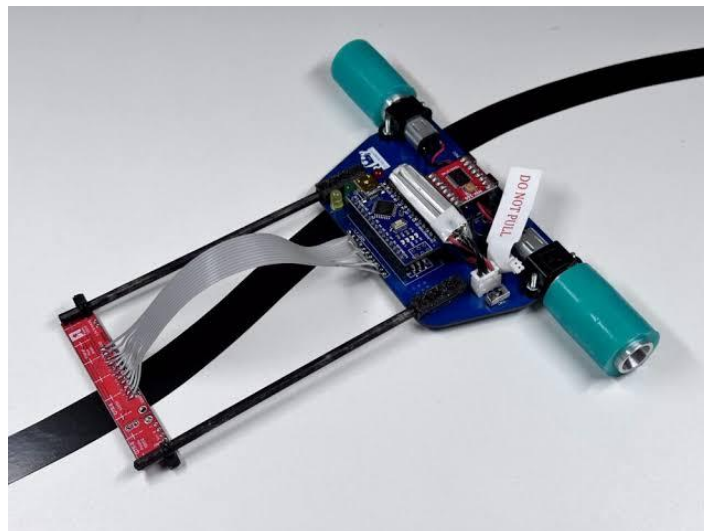


Figura 11.7: Calibración experimental de la barra de sensores sobre pista real.

11.10. Código del proyecto

El siguiente código implementa lectura de sensores, normalización, cálculo de posición y control PID. Los valores de calibración incluidos son referenciales y deben reemplazarse por los valores reales obtenidos en cada robot.

```
/*
Seguidor de linea competitivo con PID
Barra de 16 sensores multiplexados
Arduino Nano + TB6612FNG
*/

// =====
// PINES BARRA DE SENSORES
// =====
const byte S0 = A0;
const byte S1 = A1;
const byte S2 = A2;
const byte S3 = A3;
const byte OP = A7;

// =====
// PINES DRIVER TB6612FNG
// =====
const byte PWMA = 3;
const byte AIN1 = 4;
const byte AIN2 = 5;

const byte PWMB = 6;
const byte BIN1 = 7;
const byte BIN2 = 8;

const byte STBY = 9;

// =====
// PARAMETROS
// =====
const byte NUM_SENSORES = 16;
const int CENTRO_LINEA = 7500;

// Valores de calibracion referenciales
int sensorMin[NUM_SENSORES] = {
  120,120,120,120,120,120,120,120,120,
  120,120,120,120,120,120,120,120
};

int sensorMax[NUM_SENSORES] = {
```

```

850,850,850,850,850,850,850,850,
850,850,850,850,850,850,850,850
};

int lectura[NUM_SENSORES];
int lecturaNorm[NUM_SENSORES];

// Velocidad y PID
int velocidadBase = 120;
int velocidadMax = 230;

float Kp = 0.045;
float Ki = 0.0000;
float Kd = 0.75;

long error = 0;
long errorAnterior = 0;
float integral = 0;

// Recuperacion de linea
int ultimaDireccion = 0;

// =====
// MOTOR
// =====
void motorIzquierdo(int velocidad)
{
    velocidad = constrain(velocidad, -255, 255);

    if (velocidad > 0)
    {
        digitalWrite(AIN1, HIGH);
        digitalWrite(AIN2, LOW);
        analogWrite(PWMA, velocidad);
    }
    else if (velocidad < 0)
    {
        digitalWrite(AIN1, LOW);
        digitalWrite(AIN2, HIGH);
        analogWrite(PWMA, -velocidad);
    }
    else
    {
        digitalWrite(AIN1, LOW);
        digitalWrite(AIN2, LOW);
        analogWrite(PWMA, 0);
    }
}

```



```

}

void motorDerecho(int velocidad)
{
    velocidad = constrain(velocidad, -255, 255);

    if (velocidad > 0)
    {
        digitalWrite(BIN1, HIGH);
        digitalWrite(BIN2, LOW);
        analogWrite(PWMB, velocidad);
    }
    else if (velocidad < 0)
    {
        digitalWrite(BIN1, LOW);
        digitalWrite(BIN2, HIGH);
        analogWrite(PWMB, -velocidad);
    }
    else
    {
        digitalWrite(BIN1, LOW);
        digitalWrite(BIN2, LOW);
        analogWrite(PWMB, 0);
    }
}

void mover(int izquierda, int derecha)
{
    motorIzquierdo(izquierda);
    motorDerecho(derecha);
}

// =====
//  SENSORES
//  =====

void seleccionarSensor(byte canal)
{
    digitalWrite(S0, bitRead(canal, 0));
    digitalWrite(S1, bitRead(canal, 1));
    digitalWrite(S2, bitRead(canal, 2));
    digitalWrite(S3, bitRead(canal, 3));
    delayMicroseconds(40);
}

void leerSensores()
{
    for (byte i = 0; i < NUM_SENSORES; i++)

```

```

{
  seleccionarSensor(i);
  lectura[i] = analogRead(OP);

  int valor = map(lectura[i], sensorMin[i], sensorMax[i], 0, 1000);
  valor = constrain(valor, 0, 1000);

  // Si la linea es negra y entrega menor valor, invertir aqui:
  // valor = 1000 - valor;

  lecturaNorm[i] = valor;
}

long calcularPosicion()
{
  long sumaPesada = 0;
  long suma = 0;

  for (byte i = 0; i < NUM_SENSORES; i++)
  {
    long peso = (long)i * 1000;
    sumaPesada += (long)lecturaNorm[i] * peso;
    suma += lecturaNorm[i];
  }

  if (suma < 50)
  {
    // Linea perdida: girar hacia la ultima direccion conocida
    if (ultimaDireccion < 0) return 0;
    else return 15000;
  }

  long posicion = sumaPesada / suma;

  if (posicion < CENTRO_LINEA) ultimaDireccion = -1;
  if (posicion > CENTRO_LINEA) ultimaDireccion = 1;

  return posicion;
}

// =====
// PID
// =====
void controlPID()
{
  leerSensores();

```

```

long posicion = calcularPosicion();
error = posicion - CENTRO_LINEA;

integral += error;
integral = constrain(integral, -20000, 20000);

long derivada = error - errorAnterior;

float correccion = Kp * error + Ki * integral + Kd * derivada;

int pwmIzq = velocidadBase + correccion;
int pwmDer = velocidadBase - correccion;

pwmIzq = constrain(pwmIzq, -velocidadMax, velocidadMax);
pwmDer = constrain(pwmDer, -velocidadMax, velocidadMax);

mover(pwmIzq, pwmDer);

errorAnterior = error;
}

// =====
// SETUP Y LOOP
// =====
void setup()
{
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);

  pinMode(PWMA, OUTPUT);
  pinMode(AIN1, OUTPUT);
  pinMode(AIN2, OUTPUT);

  pinMode(PWMB, OUTPUT);
  pinMode(BIN1, OUTPUT);
  pinMode(BIN2, OUTPUT);

  pinMode(STBY, OUTPUT);
  digitalWrite(STBY, HIGH);

  Serial.begin(115200);
}

void loop()

```

```
{
    controlPID();
}
```

Si el robot se aleja de la línea en lugar de corregir, puede ser necesario invertir el signo de la corrección, invertir el orden físico de los sensores o intercambiar los motores izquierdo y derecho.

11.11. Ajuste experimental del PID

El ajuste de PID debe realizarse de forma progresiva. No conviene modificar todos los parámetros a la vez.

11.11.1. Ajuste de K_p

El término proporcional determina la fuerza de corrección. Si K_p es muy bajo, el robot responde tarde y se abre en curvas. Si es muy alto, oscila de lado a lado.

11.11.2. Ajuste de K_d

El término derivativo ayuda a reducir oscilaciones y anticipa cambios bruscos del error. Si K_d es demasiado bajo, el robot puede zigzaguear. Si es demasiado alto, puede reaccionar de forma agresiva o inestable.

11.11.3. Ajuste de K_i

El término integral corrige errores acumulados. En seguidores rápidos se usa con cuidado, porque puede generar acumulación excesiva y respuestas lentas. En muchos casos puede iniciar en cero.

Tabla 11.4: Guía básica para ajustar el PID.

Comportamiento observado	Posible causa	Ajuste sugerido
Se abre en curvas	K_p bajo o velocidad alta	Subir K_p o bajar velocidad
Oscila en rectas	K_p alto o K_d bajo	Bajar K_p o subir K_d
Reacciona brusco	K_d alto	Bajar K_d
Pierde línea en curvas cerradas	Velocidad alta	Bajar velocidad o agregar freno
Corrige tarde	Sensores altos o K_p bajo	Bajar sensores o subir K_p

11.12. Frenado en curvas cerradas

En curvas cerradas puede ser útil reducir la velocidad base cuando el error es grande. Esto permite que el robot entre a la curva con mayor control.

```
int calcularVelocidadBase(long error)
{
    long errorAbs = abs(error);
```

```
if (errorAbs > 4500)
{
    return 80; // curva fuerte
}
else if (errorAbs > 2500)
{
    return 100; // curva media
}
else
{
    return 130; // recta o curva suave
}
}
```

Esta función puede integrarse antes del cálculo de PWM:

```
velocidadBase = calcularVelocidadBase(error);
```

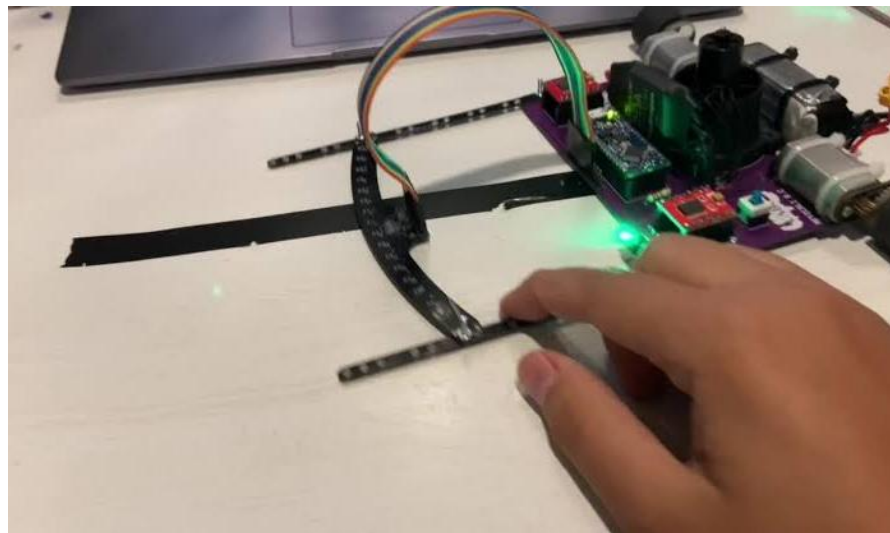


Figura 11.8: Prueba de frenado y ajuste PID en curvas cerradas.

11.13. Relación entre sensores y mecánica

El control PID no puede compensar todos los problemas mecánicos. La ubicación de sensores, altura de la barra, centro de masa, rigidez del chasis y adherencia de ruedas influyen directamente en el desempeño [7].

Recomendaciones:

- Mantener la barra de sensores paralela a la pista.
- Evitar que la barra vibre o cambie de altura.
- Colocar sensores a una distancia adecuada del eje de ruedas.
- Reducir peso innecesario.

- Usar llantas limpias y con buena adherencia.
- Verificar que ambos motores respondan de forma similar.

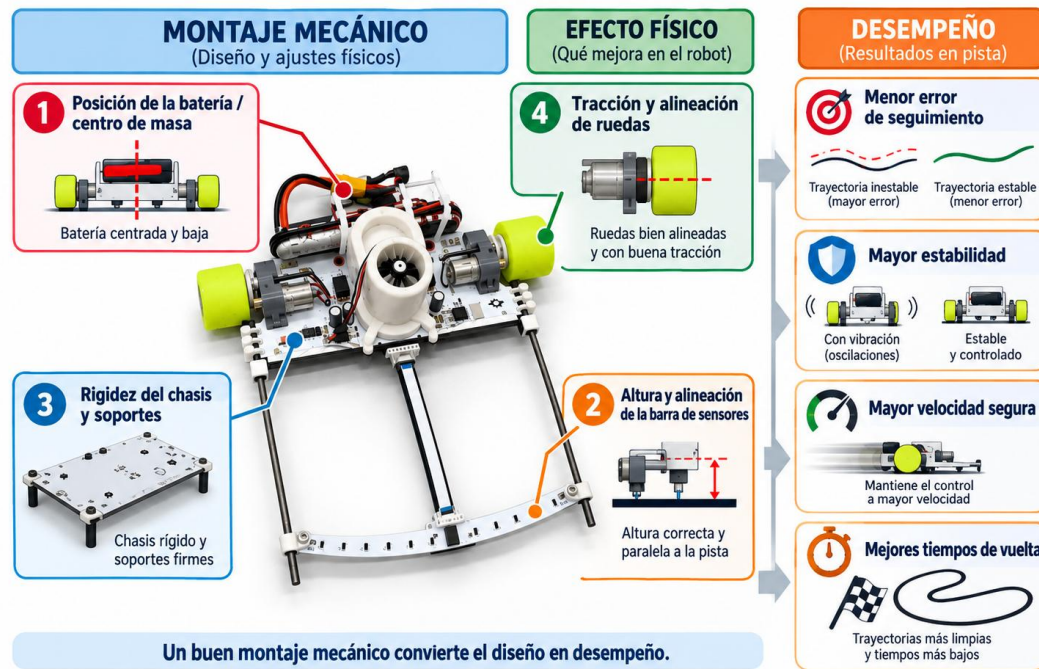


Figura 11.9: Relación entre montaje mecánico y desempeño del seguidor de línea.

11.14. Pruebas experimentales

11.14.1. Prueba 1: lectura de sensores

Ejecutar el programa de lectura cruda y verificar que todos los sensores respondan al pasar sobre línea y fondo.

11.14.2. Prueba 2: posición de línea

Mostrar por monitor serial la posición calculada y verificar que sea cercana a 7500 cuando la línea está centrada.

11.14.3. Prueba 3: seguimiento lento

Iniciar con velocidad baja y $K_i = 0$. Ajustar K_p hasta que el robot siga la línea sin perderla.

11.14.4. Prueba 4: aumento progresivo de velocidad

Aumentar velocidad base en pasos pequeños, ajustando K_d para reducir oscilación.

11.14.5. Prueba 5: curvas cerradas

Probar en curvas de 90 grados o secciones con cambios rápidos. Evaluar si se requiere reducción de velocidad.

Tabla 11.5: Registro sugerido de pruebas PID.

Prueba	Velocidad	K_p	K_d	Observación
Lento inicial				
Rectas				
Curvas suaves				
Curvas cerradas				
Pista completa				

11.15. Errores frecuentes

Durante el desarrollo pueden presentarse los siguientes errores:

- Sensores calibrados fuera de la pista real.
- Barra demasiado alta o inclinada.
- Sensor defectuoso dentro de la barra.
- Motores conectados en sentido opuesto.
- Valor de K_p demasiado alto.
- Valor de K_d demasiado bajo o excesivo.
- Velocidad base mayor que la capacidad real del robot.
- No considerar iluminación de competencia.

11.16. Proyecto de extensión

Como extensión, se propone implementar un menú de ajuste por Bluetooth para modificar velocidad, K_p , K_i y K_d sin reprogramar el robot. También se puede agregar almacenamiento en EEPROM para guardar configuraciones de diferentes pistas.

Otra mejora consiste en detectar patrones especiales, cruces o marcas laterales para cambiar de estrategia durante la carrera.

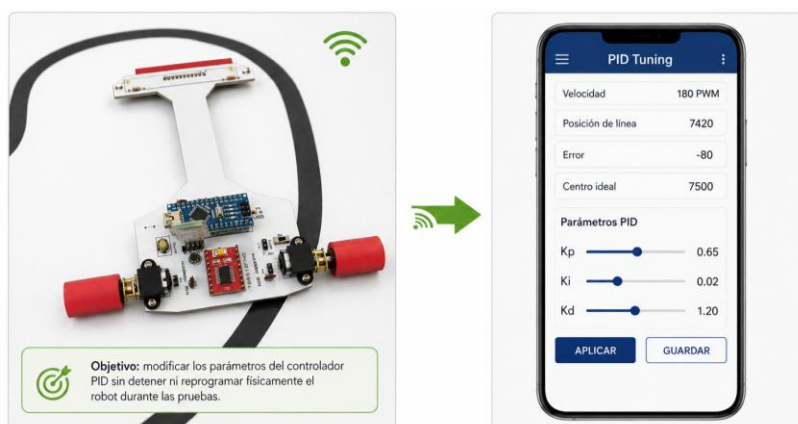


Figura 11.10: Extensión del seguidor mediante ajuste inalámbrico de parámetros PID.

Actividades de aprendizaje

1. Ejecute el programa de lectura cruda y registre valores de los 16 sensores.
2. Calibre los valores mínimos y máximos de cada sensor sobre una pista real.
3. Verifique que la posición sea cercana a 7500 cuando la línea está centrada.
4. Ajuste K_p y K_d para reducir oscilación en rectas.
5. Implemente reducción de velocidad en curvas cerradas.
6. Compare dos configuraciones de velocidad y registre cuál obtiene mejor tiempo sin salirse.

Preguntas de reflexión

1. ¿Por qué una barra de 16 sensores permite mejor control que dos sensores digitales?
2. ¿Qué ocurre si un sensor de la barra entrega valores defectuosos?
3. ¿Por qué el término derivativo ayuda a reducir la oscilación?
4. ¿Qué relación existe entre velocidad base y capacidad de corrección?
5. ¿Cómo influye la altura de la barra en la estabilidad del robot?
6. ¿Qué cambios realizaría si el robot se abre demasiado al salir de una recta?

Referencias

- [1] Pololu, “QTR Reflectance Sensor Application Note”, Pololu Documentation. [En línea]. Disponible en: <https://www.pololu.com/docs/0J13>, [Accedido: 24-jun-2026].
- [2] Arduino, “analogRead()”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/language-reference/en/functions/analog-io/analogRead/>. [Accedido: 24-jun-2026].
- [3] Arduino, “analogWrite()”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/language-reference/en/functions/analog-io/analogWrite/>. [Accedido: 24-jun-2026].
- [4] Toshiba Electronic Devices & Storage Corporation, “TB6612FNG Brush Motor Driver IC”, Toshiba. [En línea]. Disponible en: <https://toshiba.semicon-storage.com/ap-en/semiconductor/product/motor-driver-ics/brushed-dc-motor-driver-ics/detail.TB6612FNG.htm>. [Accedido: 24-jun-2026].
- [5] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ, USA: Princeton University Press, 2008.
- [6] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.
- [7] D. Cook, *Robot Building for Beginners*, 4th ed. Berkeley, CA, USA: Apress, 2020.

Capítulo 12. PID Web Tuning con ESP32

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender la utilidad del ajuste inalámbrico de parámetros en robots competitivos.
- Configurar un ESP32 como punto de acceso WiFi para alojar un panel web local.
- Crear una interfaz web básica para modificar K_p , K_i , K_d , velocidad base y velocidad máxima.
- Implementar un servidor web embebido en ESP32 usando la librería WebServer.
- Integrar parámetros ajustables en un controlador PID para motores diferenciales.
- Guardar parámetros en memoria no volátil mediante Preferences.
- Aplicar el sistema a un seguidor de línea, robot móvil o banco de pruebas de control.

12.1. Introducción

El ajuste de parámetros es una de las tareas más importantes en robots competitivos. En un seguidor de línea, pequeños cambios en K_p , K_d o velocidad base pueden marcar la diferencia entre un recorrido estable y una salida de pista. En un Minisumo, la velocidad de búsqueda, el tiempo de evasión o la potencia de ataque pueden variar según el dohyo, la iluminación y el oponente. En una plataforma móvil, los parámetros de control deben adaptarse a peso, fricción, batería y superficie.

Tradicionalmente, el ajuste se realiza modificando el código, compilando, cargando nuevamente el programa y probando el robot. Este ciclo consume tiempo, especialmente durante competencias donde el margen para realizar cambios es limitado. Una solución práctica consiste en incorporar una interfaz de ajuste en tiempo real, permitiendo modificar parámetros desde un celular o computadora sin conectar cable USB ni reprogramar el microcontrolador.

En este capítulo se desarrolla un sistema denominado PID Web Tuning con ESP32. El proyecto utiliza un ESP32 configurado como punto de acceso WiFi. El usuario se conecta desde un navegador web y accede a un panel donde puede modificar K_p , K_i , K_d , velocidad base y velocidad máxima. Los nuevos valores se aplican al robot y pueden guardarse en memoria no volátil.

El proyecto se plantea como una extensión del Capítulo 11, donde se implementó un seguidor de línea competitivo con PID. Sin embargo, la misma metodología puede aplicarse a otros robots que requieran ajuste de parámetros [6].

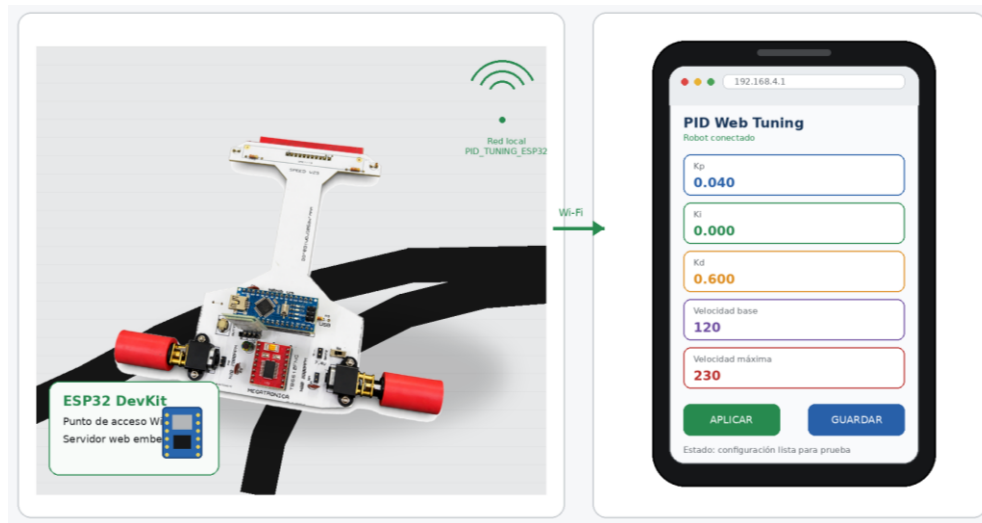


Figura 12.1: Ajuste inalámbrico de parámetros PID desde un navegador web.

12.2. Planteamiento del problema

Durante la puesta a punto de un robot seguidor de línea, el programador debe realizar múltiples pruebas:

- Aumentar o reducir K_p .
- Ajustar K_d para reducir oscilaciones.
- Probar si K_i mejora o perjudica la respuesta.
- Cambiar la velocidad base.
- Limitar la velocidad máxima.
- Ajustar frenos en curvas cerradas.

Si cada cambio requiere conectar el robot a la computadora, modificar el código y cargar nuevamente el programa, el proceso se vuelve lento. Además, durante competencias no siempre es cómodo o posible reprogramar el robot junto a la pista.

El sistema propuesto permite modificar parámetros desde una interfaz web local. El ESP32 crea su propia red WiFi, de modo que no se requiere router, internet ni infraestructura externa.

El ajuste inalámbrico no reemplaza el buen diseño del controlador. Su función es acelerar el proceso experimental y permitir comparar configuraciones de forma rápida y ordenada.

12.3. Descripción general del proyecto

El proyecto utiliza un ESP32 como controlador principal del robot o como módulo auxiliar de configuración. En la versión presentada, el ESP32 cumple las dos funciones: ejecuta el control PID y aloja el panel web [7].

El sistema se compone de:

- ESP32 DevKit.
- Barra de sensores o entrada de error simulada.
- Driver de motores. Dos motores DC.
- Servidor web embebido.
- Memoria no volátil para guardar parámetros.
- Celular o computadora conectada por WiFi.

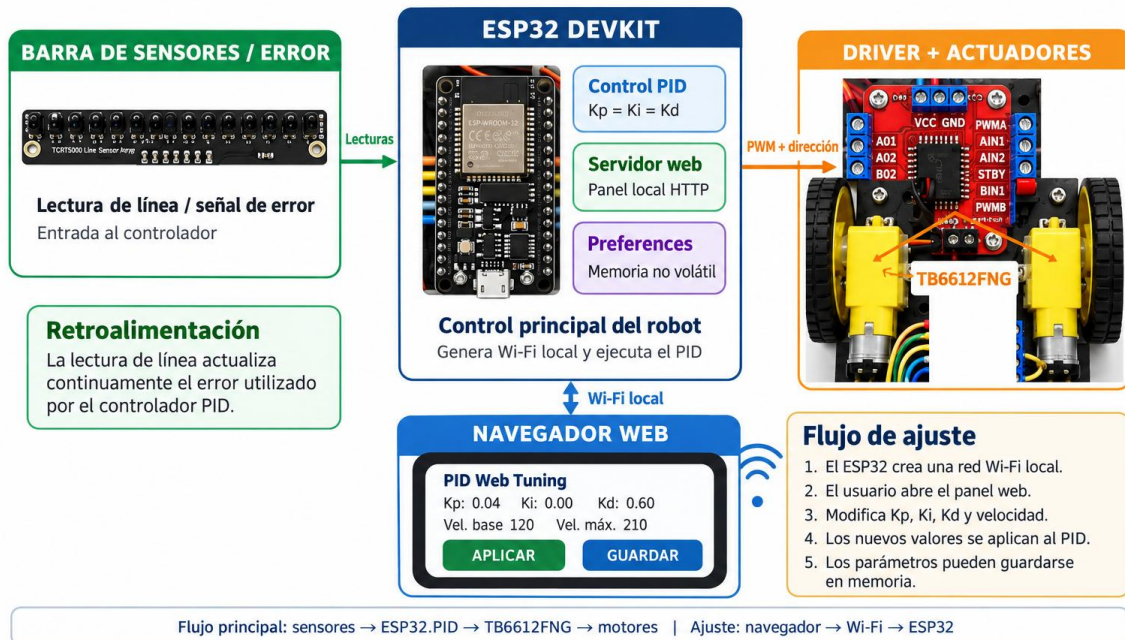


Figura 12.2: Arquitectura general del sistema PID Web Tuning.

12.4. Aplicación STEM

Este proyecto integra las áreas STEM de la siguiente forma:

- **Ciencia:** análisis de respuesta dinámica, estabilidad, oscilación y corrección de error.
- **Tecnología:** uso de ESP32, WiFi, servidor web embebido y memoria no volátil.
- **Ingeniería:** diseño de una herramienta de ajuste para mejorar robots reales.
- **Matemáticas:** parámetros PID, error, derivada, integral, saturación y comparación experimental.

12.5. Objetivos del proyecto

12.5.1. Objetivo general

Diseñar e implementar un sistema de ajuste inalámbrico de parámetros PID mediante ESP32 y una interfaz web local, aplicable a robots seguidores de línea y plataformas móviles.

12.5.2. Objetivos específicos

- 1. Configurar el ESP32 como punto de acceso WiFi.
- 2. Crear un servidor web embebido con formularios de ajuste.
- 3. Modificar K_p , K_i , K_d , velocidad base y velocidad máxima desde el navegador.
- 4. Guardar parámetros en memoria no volátil.
- 5. Aplicar los parámetros al control PID del robot.
- 6. Evaluar experimentalmente diferentes configuraciones.

12.6. Materiales y herramientas

Tabla 12.1: Materiales principales para PID Web Tuning.

Elemento	Cantidad	Función
ESP32 DevKit	1	Control, WiFi y servidor web
Driver TB6612FNG o similar	1	Control de motores DC
Motores DC con caja reductora	2	Tracción diferencial
Barra de sensores o sensor de prueba	1	Medición de línea o error
Batería LiPo	1	Alimentación del robot
Regulador DC-DC	1	Alimentación estable del ESP32
Celular o computadora	1	Interfaz web de ajuste
Pista o banco de pruebas	1	Validación experimental



Figura 12.3: Materiales utilizados para implementar PID Web Tuning.

12.7. Fundamento teórico

12.7.1. ESP32 como punto de acceso

El ESP32 puede operar en modo estación, punto de acceso o modo mixto. En modo punto de acceso, el propio ESP32 crea una red WiFi a la que se puede conectar un celular o computadora.

Esta característica es útil cuando el robot se prueba en laboratorios, pistas o competencias sin depender de una red externa.

La documentación de Arduino-ESP32 incluye soporte para WiFi mediante la librería WiFi, permitiendo configurar el ESP32 como punto de acceso usando WiFi.softAP() [1, 2].

12.7.2. Servidor web embebido

Un servidor web embebido permite que el ESP32 responda a solicitudes HTTP. El usuario abre una dirección IP en el navegador y recibe una página HTML generada por el microcontrolador [3].

En este proyecto, la página web tendrá:

- Campos para K_p , K_i , K_d .
- Campo para velocidad base.
- Campo para velocidad máxima.
- Botón para aplicar cambios.
- Botón para guardar parámetros.
- Visualización de valores actuales.

12.7.3. Memoria no volátil con Preferences

La librería Preferences permite almacenar datos en memoria no volátil del ESP32. Esto significa que los parámetros pueden conservarse aunque el robot se apague. La documentación de Arduino-ESP32 presenta esta librería como una forma de guardar pares clave-valor en la memoria no volátil del ESP32 [2].

12.7.4. Parámetros PID ajustables

El controlador PID utilizado en el Capítulo 12 puede adaptarse para que sus constantes no sean fijas en el código, sino variables modificables desde la interfaz web [4, 5].

$$u = K_p e + K_i \sum e + K_d (e - e_{anterior})$$

La ventaja de este enfoque es que permite probar diferentes combinaciones sin recompilar el programa.

12.8. Diseño electrónico

En esta versión se propone un robot con ESP32 como controlador principal. Para simplificar, la lectura de sensores puede adaptarse según la barra utilizada. El enfoque del capítulo se centra en la interfaz web y el ajuste de parámetros.

Tabla 12.2: Asignación sugerida de pines del ESP32 para motores y sensores.

Pin ESP32	Conexión	Función
GPIO32	Sensor / error analógico	Entrada de prueba o sensor central
GPIO25	PWMA driver	PWM motor izquierdo
GPIO26	AIN1 driver	Dirección motor izquierdo
GPIO27	AIN2 driver	Dirección motor izquierdo
GPIO14	PWMB driver	PWM motor derecho
GPIO12	BIN1 driver	Dirección motor derecho
GPIO13	BIN2 driver	Dirección motor derecho
GPIO33	STBY driver	Habilitación driver
GND	GND común	Referencia del sistema

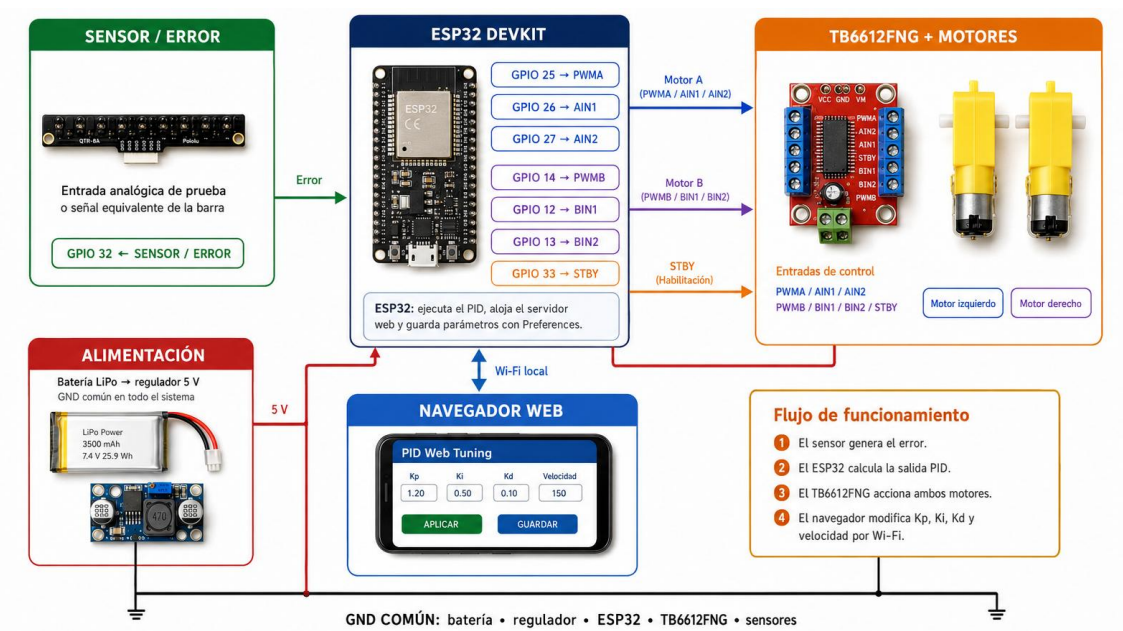


Figura 12.4: Diagrama de conexiones del sistema PID Web Tuning.

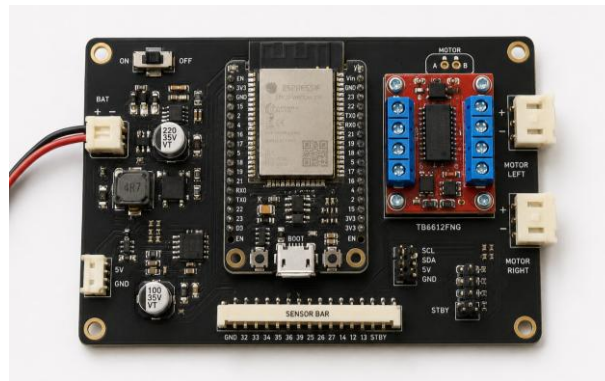


Figura 12.5: Montaje físico del sistema PID Web Tuning.

12.9. Diseño de la interfaz web

La interfaz web debe ser simple y rápida. Durante una competencia o prueba, el usuario no necesita una página compleja, sino controles claros.

Elementos sugeridos:

- Título del robot.
- Valores actuales de K_p , K_i , K_d .
- Velocidad base.
- Velocidad máxima.
- Botón Aplicar.
- Botón Guardar.
- Estado del robot



Figura 12.6: Interfaz web para ajuste inalámbrico de parámetros PID.

12.10. Código del proyecto

El siguiente código crea una red WiFi local, aloja una página web y permite modificar parámetros PID. Para que el ejemplo sea claro, se incluye una entrada analógica simple como error de prueba. El lector puede reemplazar esa función por el cálculo de posición de línea desarrollado en el Capítulo 11.

```
#include <WiFi.h>
#include <WebServer.h>
#include <Preferences.h>
```

```

// =====
// WIFI EN MODO PUNTO DE ACCESO
// =====
const char* ssid = "PID_TUNING_ESP32";
const char* password = "12345678";

WebServer server(80);
Preferences prefs;

// =====
// PINES DE MOTOR
// =====
const byte SENSOR_ERROR = 32;

const byte PWMA = 25;
const byte AIN1 = 26;
const byte AIN2 = 27;

const byte PWMB = 14;
const byte BIN1 = 12;
const byte BIN2 = 13;

const byte STBY = 33;

// =====
// PWM ESP32
// =====
const byte CANAL_A = 0;
const byte CANAL_B = 1;
const int FRECUENCIA_PWM = 20000;
const byte RESOLUCION_PWM = 8;

// =====
// PARAMETROS AJUSTABLES
// =====
float Kp = 0.04;
float Ki = 0.00;
float Kd = 0.60;

int velocidadBase = 120;
int velocidadMax = 230;

long error = 0;
long errorAnterior = 0;
float integral = 0;

```



```

// =====
// CONTROL DE MOTORES
// =====
void motorIzquierdo(int velocidad)
{
    velocidad = constrain(velocidad, -255, 255);

    if (velocidad > 0)
    {
        digitalWrite(AIN1, HIGH);
        digitalWrite(AIN2, LOW);
        ledcWrite(CANAL_A, velocidad);
    }
    else if (velocidad < 0)
    {
        digitalWrite(AIN1, LOW);
        digitalWrite(AIN2, HIGH);
        ledcWrite(CANAL_A, -velocidad);
    }
    else
    {
        digitalWrite(AIN1, LOW);
        digitalWrite(AIN2, LOW);
        ledcWrite(CANAL_A, 0);
    }
}

void motorDerecho(int velocidad)
{
    velocidad = constrain(velocidad, -255, 255);

    if (velocidad > 0)
    {
        digitalWrite(BIN1, HIGH);
        digitalWrite(BIN2, LOW);
        ledcWrite(CANAL_B, velocidad);
    }
    else if (velocidad < 0)
    {
        digitalWrite(BIN1, LOW);
        digitalWrite(BIN2, HIGH);
        ledcWrite(CANAL_B, -velocidad);
    }
    else
    {
        digitalWrite(BIN1, LOW);
        digitalWrite(BIN2, LOW);
    }
}

```

```

    ledcWrite(CANAL_B, 0);
}
}

void mover(int izquierda, int derecha)
{
    motorIzquierdo(izquierda);
    motorDerecho(derecha);
}

// =====
// ERROR DE PRUEBA
// Reemplazar por calculo real de posicion de linea
// =====
long leerError()
{
    int valor = analogRead(SENSOR_ERROR);
    long e = map(valor, 0, 4095, -7500, 7500);
    return e;
}

// =====
// CONTROL PID
// =====
void controlPID()
{
    error = leerError();

    integral += error;
    integral = constrain(integral, -20000, 20000);

    long derivada = error - errorAnterior;

    float correccion = Kp * error + Ki * integral + Kd * derivada;

    int pwmIzq = velocidadBase + correccion;
    int pwmDer = velocidadBase - correccion;

    pwmIzq = constrain(pwmIzq, -velocidadMax, velocidadMax);
    pwmDer = constrain(pwmDer, -velocidadMax, velocidadMax);
    mover(pwmIzq, pwmDer);

    errorAnterior = error;
}

// =====
// MEMORIA NO VOLATIL

```

```

// =====
void cargarParametros()
{
    prefs.begin("pid", true);

    Kp = prefs.getFloat("kp", Kp);
    Ki = prefs.getFloat("ki", Ki);
    Kd = prefs.getFloat("kd", Kd);
    velocidadBase = prefs.getInt("vbase", velocidadBase);
    velocidadMax = prefs.getInt("vmax", velocidadMax);

    prefs.end();
}

void guardarParametros()
{
    prefs.begin("pid", false);

    prefs.putFloat("kp", Kp);
    prefs.putFloat("ki", Ki);
    prefs.putFloat("kd", Kd);
    prefs.putInt("vbase", velocidadBase);
    prefs.putInt("vmax", velocidadMax);

    prefs.end();
}

// =====
// PAGINA WEB
// =====
String paginaHTML()
{
    String html = "";
    html += "<!DOCTYPE html><html><head>";
    html += "<meta name='viewport' content='width=device-width,initial-scale=1'>";
    html += "<title>PID Web Tuning</title>";
    html += "<style>";
    html +=
        "body{font-family:Arial;background:#111;color:#eee;text-align:center;}";
    html += ".card{background:#222;margin:20px
        auto;padding:20px;border-radius:12px;max-width:420px;}";
    html += "input{width:90 %;padding:10px;margin:8px;font-size:16px;}";
    html += "button{padding:12px
        20px;margin:8px;font-size:16px;border:0;border-radius:8px;}";
    html += "</style></head><body>";
    html += "<div class='card'>";
    html += "<h2>PID Web Tuning</h2>";

```

```

html += "<form action='/set' method='GET'>";
html += "Kp:<input name='kp' value='" + String(Kp, 5) + "'><br>";
html += "Ki:<input name='ki' value='" + String(Ki, 5) + "'><br>";
html += "Kd:<input name='kd' value='" + String(Kd, 5) + "'><br>";
html += "Velocidad base:<input name='vbase' value='" + String(velocidadBase)
+ "'><br>";
html += "Velocidad maxima:<input name='vmax' value='" + String(velocidadMax)
+ "'><br>";
html += "<button type='submit'>Aplicar</button>";
html += "</form>";
html += "<form action='/save' method='GET'>";
html += "<button type='submit'>Guardar en memoria</button>";
html += "</form>";
html += "<p>Error actual: " + String(error) + "</p>";
html += "</div></body></html>";

return html;
}

void manejarRaiz()
{
    server.send(200, "text/html", paginaHTML());
}

void manejarSet()
{
    if (server.hasArg("kp")) Kp = server.arg("kp").toFloat();
    if (server.hasArg("ki")) Ki = server.arg("ki").toFloat();
    if (server.hasArg("kd")) Kd = server.arg("kd").toFloat();
    if (server.hasArg("vbase")) velocidadBase = server.arg("vbase").toInt();
    if (server.hasArg("vmax")) velocidadMax = server.arg("vmax").toInt();

    velocidadBase = constrain(velocidadBase, 0, 255);
    velocidadMax = constrain(velocidadMax, 0, 255);

    server.sendHeader("Location", "/");
    server.send(303);
}

void manejarSave()
{
    guardarParametros();

    server.send(200, "text/html",
"<html><body><h2>Parametros guardados</h2><a
href='/'>Volver</a></body></html>");
}

```

```

// =====
// SETUP Y LOOP
// =====
void setup()
{
  Serial.begin(115200);

  pinMode(AIN1, OUTPUT);
  pinMode(AIN2, OUTPUT);
  pinMode(BIN1, OUTPUT);
  pinMode(BIN2, OUTPUT);
  pinMode(STBY, OUTPUT);

  digitalWrite(STBY, HIGH);

  ledcSetup(CANAL_A, FRECUENCIA_PWM, RESOLUCION_PWM);
  ledcSetup(CANAL_B, FRECUENCIA_PWM, RESOLUCION_PWM);

  ledcAttachPin(PWMA, CANAL_A);
  ledcAttachPin(PWMB, CANAL_B);

  cargarParametros();

  WiFi.softAP(ssid, password);

  Serial.println("Red WiFi creada");
  Serial.print("SSID: ");
  Serial.println(ssid);
  Serial.print("IP: ");
  Serial.println(WiFi.softAPIP());

  server.on("/", manejarRaiz);
  server.on("/set", manejarSet);
  server.on("/save", manejarSave);

  server.begin();
}

void loop()
{
  server.handleClient();
  controlPID();
}

```

El código usa una lectura analógica como error de prueba. Para un seguidor de línea real, la función leerError() debe reemplazarse por el cálculo de posición de línea y error desarrollado en el Capítulo 11.

12.11. Integración con el seguidor de línea

Para integrar este sistema con un seguidor de línea real, se debe reemplazar:

```
long leerError()
{
  int valor = analogRead(SENSOR_ERROR);
  long e = map(valor, 0, 4095, -7500, 7500);
  return e;
}
```

por una función basada en la barra de sensores:

```
long leerError()
{
  leerSensores();
  long posicion = calcularPosicion();
  return posicion - CENTRO_LINEA;
}
```

De esta manera, el panel web controla los parámetros reales del robot.

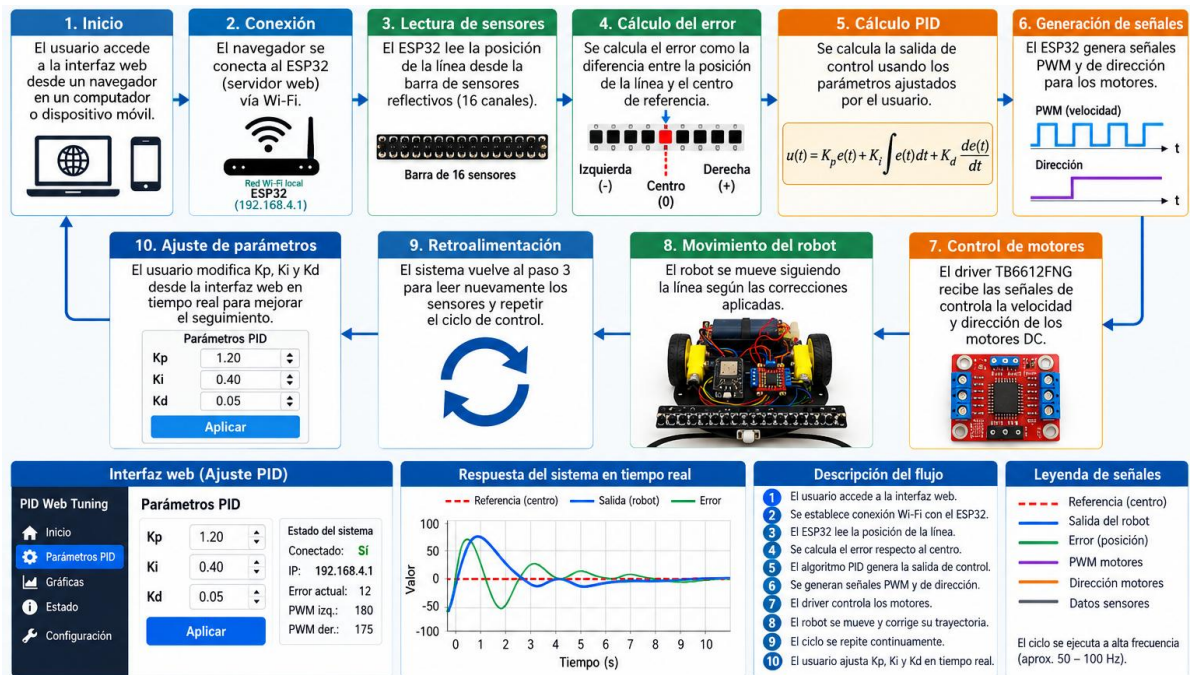


Figura 12.7: Integración de PID Web Tuning en un robot seguidor de línea.

12.12. Procedimiento de uso

1. Cargar el programa en el ESP32.
2. Encender el robot.
3. Buscar la red WiFi PID_TUNING_ESP32 desde el celular.
4. Conectarse usando la contraseña 12345678.
5. Abrir en el navegador la dirección IP mostrada en el monitor serial.
6. Modificar K_p , K_i , K_d , velocidad base y velocidad máxima.
7. Presionar *Aplicar*.
8. Probar el robot en pista.
9. Presionar Guardar en memoria si la configuración es adecuada.

12.13. Tabla de pruebas

Se recomienda registrar cada combinación probada. Esto evita repetir ajustes y permite comparar resultados objetivamente.

Tabla 12.3: Registro sugerido para pruebas de PID Web Tuning.

Prueba	K_p	K_i	K_d	Vel.	Observación
1					
2					
3					
4					
5					

12.14. Criterios de ajuste

12.14.1. Ajuste inicial

Se recomienda iniciar con velocidad baja, $K_i = 0$, K_d moderado y K_p pequeño. A partir de ahí, aumentar progresivamente K_p hasta que el robot siga la línea con respuesta clara.

12.14.2. Control de oscilación

Si el robot zigzaguea en rectas, puede reducirse K_p o aumentar K_d . Si el robot responde lento en curvas, puede aumentarse K_p , reducir velocidad o mejorar la posición de sensores.

12.14.3. Velocidad base

La velocidad base debe incrementarse solo cuando el robot ya sigue la línea de forma estable. Aumentar velocidad antes de ajustar el control suele producir salidas de pista.

12.14.4. Uso del término integral

El término integral debe usarse con cuidado. En seguidores de línea rápidos, un K_i elevado puede acumular error y provocar correcciones tardías. Se recomienda iniciar con $K_i = 0$ y aumentarlo solo si existe un error persistente.

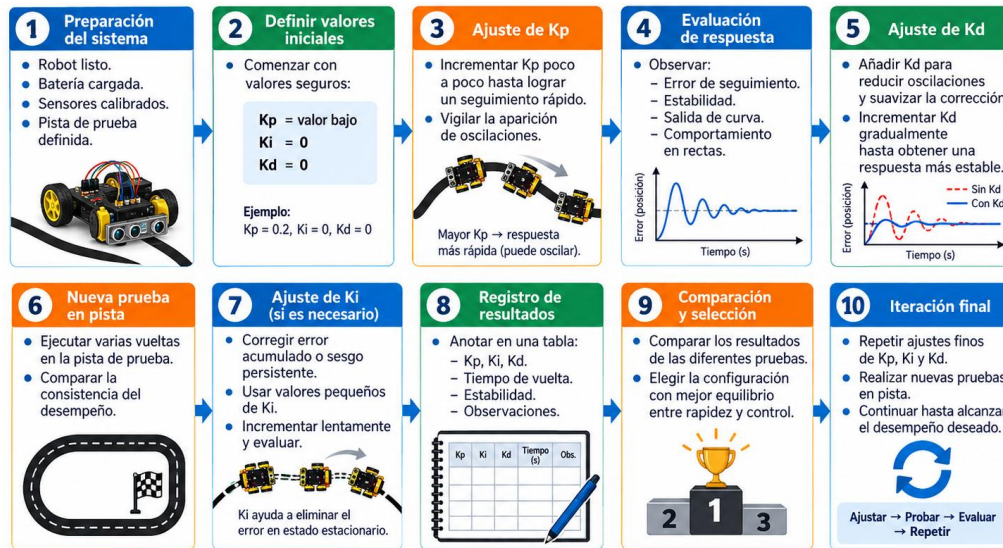


Figura 12.8: Proceso recomendado de ajuste experimental del PID.

12.15. Seguridad durante pruebas

Aunque el ajuste web permite modificar parámetros rápidamente, esto también puede provocar comportamientos inesperados. Por ello se recomienda:

- Probar inicialmente con el robot elevado.
- Limitar velocidad máxima durante las primeras pruebas.
- Mantener un interruptor físico de apagado.
- No modificar parámetros extremos mientras el robot está en alta velocidad.
- Verificar batería antes de pruebas largas.
- Registrar qué valores producen comportamientos inestables.

Un valor demasiado alto de K_p , K_d o velocidad puede hacer que el robot oscile violentamente o se salga de la pista. El panel web facilita el ajuste, pero no elimina la necesidad de pruebas seguras.

12.16. Errores frecuentes

Durante el desarrollo pueden presentarse los siguientes errores:

- No conectarse a la red WiFi creada por el ESP32.
- Escribir una dirección IP incorrecta.

- Usar una contraseña menor de ocho caracteres para el punto de acceso.
- No llamar a `server.handleClient()` dentro del `loop()`.
- Guardar parámetros inestables en memoria.
- No limitar velocidad máxima.
- Usar pines PWM inadecuados o mal configurados.
- Alimentar el ESP32 desde una fuente inestable.

12.17. Proyecto de extensión

Como extensión, se propone agregar una gráfica en tiempo real del error de línea, utilizando peticiones periódicas desde la página web. También puede incorporarse un botón de parada, selección de perfiles de pista y almacenamiento de múltiples configuraciones.

Otra mejora consiste en separar el sistema en dos microcontroladores: un Arduino Nano dedicado al control rápido del robot y un ESP32 encargado únicamente de la interfaz web. En ese caso, ambos pueden comunicarse mediante UART o I2C.

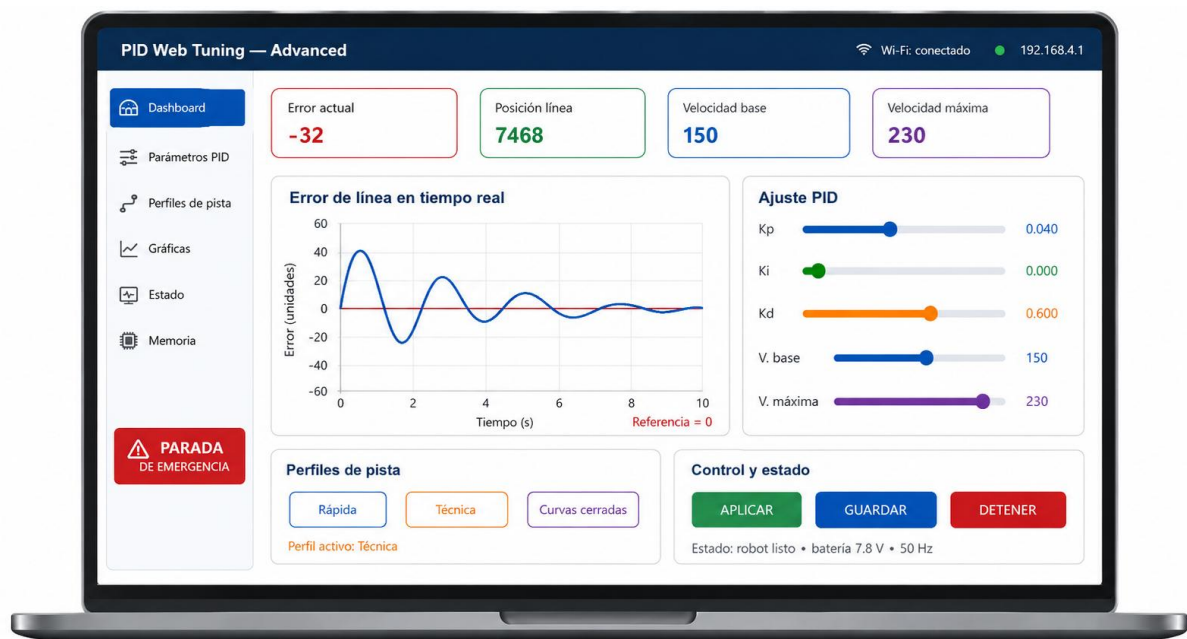


Figura 12.9: Extensión avanzada del panel PID Web Tuning.

Actividades de aprendizaje

1. Cargue el programa en un ESP32 y verifique la creación de la red WiFi.
2. Acceda al panel web desde un celular.
3. Modifique K_p , K_i , K_d , y observe los cambios en el monitor serial.
4. Integre la función `leerError()` con una barra de sensores real.

5. Guarde una configuración en memoria y verifique que permanezca después de reiniciar.
6. Diseñe una versión del panel con botones para perfiles de pista.

Preguntas de reflexión

1. ¿Qué ventaja ofrece ajustar PID desde una interfaz web durante pruebas?
2. ¿Por qué es útil que el ESP32 cree su propia red WiFi?
3. ¿Qué riesgos existen al modificar parámetros mientras el robot está en movimiento?
4. ¿Por qué conviene guardar solo configuraciones verificadas?
5. ¿Qué beneficios tendría separar el control del robot y la interfaz web en dos microcontroladores?

Referencias

- [1] Espressif Systems, “Wi-Fi API”, Arduino ESP32 Documentation. [En línea]. Disponible en: <https://docs.espressif.com/projects/arduino-esp32/en/latest/api/wifi.html>. [Accedido: 24-jun-2026].
- [2] Espressif Systems, “Preferences”, Arduino ESP32 Documentation. [En línea]. Disponible en: <https://docs.espressif.com/projects/arduino-esp32/en/latest/api/preferences.html>. [Accedido: 24-jun-2026].
- [3] Espressif Systems, “WebServer library for Arduino ESP32”, Arduino ESP32 Documentation. [En línea]. Disponible en: <https://docs.espressif.com/projects/arduino-esp32/en/latest/api/webserver.html>. [Accedido: 24-jun-2026].
- [4] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ, USA: Princeton University Press, 2008.
- [5] K. Ogata, *Modern Control Engineering*, 5th ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010.
- [6] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.
- [7] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2017.

Capítulo 13. Visión Artificial con ESP32-CAM y OpenCV

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender el papel de la visión artificial en robots móviles y sistemas inteligentes.
- Configurar una ESP32-CAM para transmitir video mediante WiFi.
- Capturar imágenes desde una computadora utilizando Python y OpenCV.
- Procesar imágenes en el espacio de color HSV para detectar objetos por color.
- Calcular el centroide de un objeto detectado y clasificar su posición en izquierda, centro o derecha.
- Relacionar la detección visual con decisiones de control para robots móviles.
- Identificar limitaciones asociadas a iluminación, resolución, latencia y velocidad de procesamiento.

13.1. Introducción

La visión artificial permite que un sistema embebido o computacional interprete información visual del entorno. En robótica, esta capacidad puede utilizarse para detectar objetos, reconocer colores, identificar marcas, seguir trayectorias, localizar oponentes, inspeccionar piezas o asistir en tareas de navegación. Aunque existen cámaras industriales y procesadores especializados, también es posible introducir estos conceptos con herramientas accesibles como ESP32-CAM, Python y OpenCV [8].

La ESP32-CAM es una placa compacta que integra un microcontrolador ESP32, cámara OV2640 y conectividad WiFi. Esto permite transmitir imágenes de manera inalámbrica hacia una computadora, celular o sistema de procesamiento. Por su bajo costo y tamaño reducido, resulta útil para proyectos educativos, demostraciones de visión, robots exploradores, monitoreo básico y prototipos de interacción [2].

En este capítulo se desarrolla un proyecto de visión artificial basado en dos etapas. En la primera, la ESP32-CAM se configura como cámara WiFi para enviar video dentro de una red local. En la segunda, una computadora ejecuta un programa en Python con OpenCV para capturar la imagen, detectar un objeto de color específico, calcular su posición y generar una decisión de control.

El enfoque del capítulo es práctico y educativo. No se busca desarrollar un sistema de visión industrial, sino introducir una metodología aplicable a robots móviles, seguidores con visión, robots exploradores, prototipos de clasificación por color y demostraciones del Club de Robótica UTE.

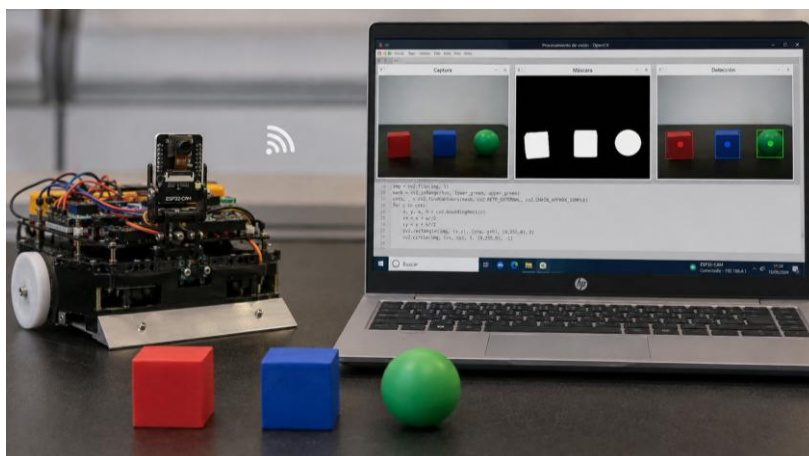


Figura 13.1: Sistema de visión artificial con ESP32-CAM y procesamiento en computadora.

13.2. Planteamiento del problema

Los robots competitivos y educativos suelen utilizar sensores de distancia, sensores reflectivos o interruptores para percibir el entorno. Estos sensores son rápidos y económicos, pero ofrecen información limitada. Una cámara puede proporcionar más información: forma, color, posición relativa, tamaño aparente y movimiento.

Sin embargo, incorporar visión artificial presenta varios desafíos:

- La imagen cambia con la iluminación.
- El procesamiento puede generar latencia.
- La detección por color puede fallar con fondos similares.
- La cámara requiere buena orientación mecánica.
- La resolución afecta velocidad y precisión.
- El sistema debe convertir información visual en decisiones simples.

Para iniciar de forma controlada, se propone detectar un objeto de color específico y clasificar su posición horizontal dentro de la imagen. Esta información puede utilizarse para decidir si un robot debe girar a la izquierda, avanzar o girar a la derecha.

La visión artificial no consiste solo en mostrar una cámara. El objetivo es extraer información útil de la imagen y convertirla en una decisión para el robot.

13.3. Descripción general del proyecto

El sistema propuesto consta de dos partes:

- **Módulo de captura:** ESP32-CAM transmite imágenes mediante WiFi.
- **Módulo de procesamiento:** computadora con Python y OpenCV detecta el objeto por color.

La computadora recibe el video, convierte cada imagen al espacio de color HSV, aplica un umbral de color, obtiene una máscara binaria, identifica contornos y calcula el centroide del objeto principal.

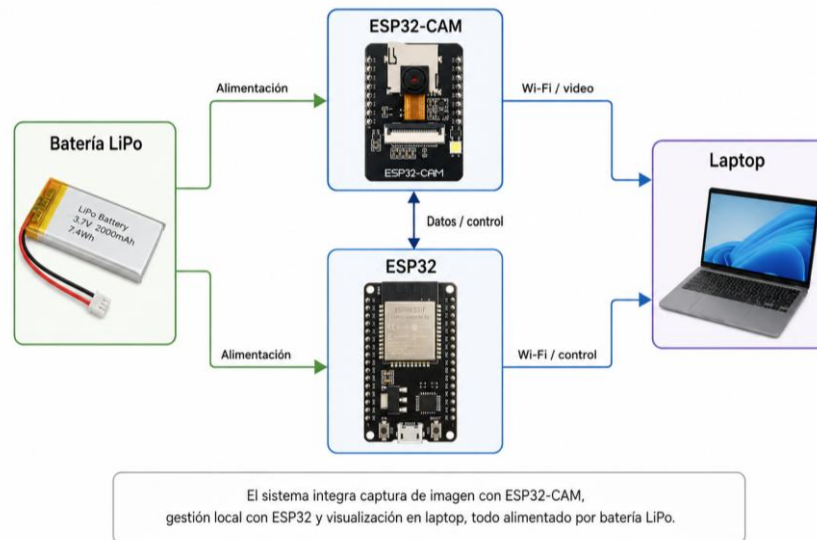


Figura 13.2: Arquitectura general del sistema de visión artificial.

13.4. Aplicación STEM

Este proyecto integra las áreas STEM de la siguiente manera:

- **Ciencia:** estudio de luz, color, reflexión, percepción visual y variación por iluminación.
- **Tecnología:** uso de ESP32-CAM, WiFi, Python, OpenCV y procesamiento digital de imágenes.
- **Ingeniería:** diseño de una solución de percepción visual para robots móviles.
- **Matemáticas:** umbrales, coordenadas, centroide, áreas, segmentación y clasificación de regiones.

13.5. Objetivos del proyecto

13.5.1. Objetivo general

Diseñar e implementar un sistema de visión artificial con ESP32-CAM y OpenCV para detectar un objeto por color, localizar su posición en la imagen y generar decisiones básicas de control.

13.5.2. Objetivos específicos

1. Configurar la ESP32-CAM como cámara WiFi.
2. Capturar el video de la ESP32-CAM desde Python.
3. Convertir imágenes del espacio BGR al espacio HSV.

4. Aplicar segmentación por color mediante umbrales.
5. Calcular contornos y centroide del objeto detectado.
6. Clasificar la posición del objeto como izquierda, centro o derecha.
7. Proponer la integración del resultado con un robot móvil.

13.6. Materiales y herramientas

Tabla 13.1: Materiales principales para el sistema de visión artificial.

Elemento	Cantidad	Función
ESP32-CAM AI Thinker o similar	1	Captura y transmisión de imagen
Módulo USB-TTL	1	Programación de la ESP-CAM
Computadora con Python	1	Procesamiento de imagen
Red WiFi o punto de acceso	1	Comunicación entre cámara y computadora
Objeto de color definido	1 o más	Elemento a detectar
Robot móvil o banco de pruebas	1	Plataforma de aplicación
Fuente de 5 V estable	1	Alimentación de ESP32-CAM
Arduino IDE	1	Programación del módulo
OpenCV	1	Procesamiento de imágenes

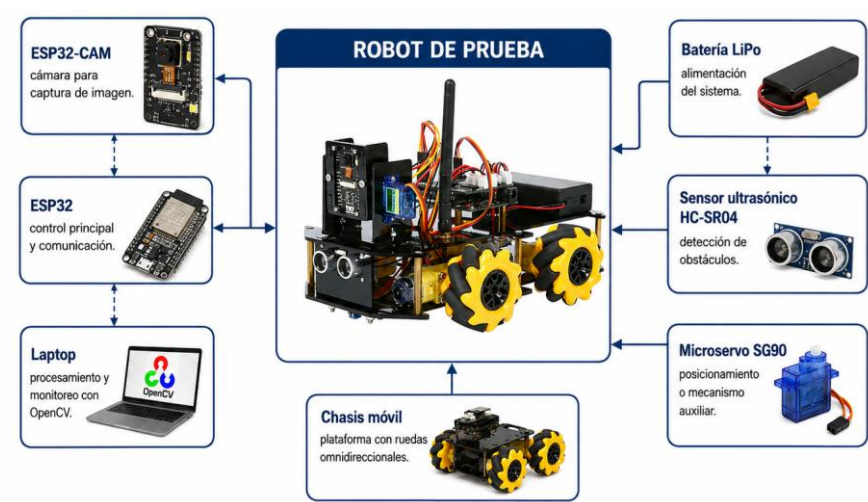


Figura 13.3: Materiales utilizados para el proyecto de visión artificial.

13.7. Fundamento teórico

13.7.1. ESP32-CAM

La ESP32-CAM integra un microcontrolador ESP32 y una cámara, comúnmente OV2640. El entorno Arduino-ESP32 proporciona soporte para cámaras mediante la librería `esp_camera`, utilizada en ejemplos como `CameraWebServer` [1]. Esto permite configurar pines, resolución, calidad JPEG y transmisión de imágenes.

La placa suele requerir un módulo USB-TTL para programación, ya que muchos modelos no incluyen conversor USB integrado. También requiere una fuente de alimentación estable, especialmente durante transmisión WiFi.

13.7.2. OpenCV

OpenCV es una biblioteca ampliamente utilizada para visión por computadora. Permite leer imágenes, procesarlas, detectar contornos, aplicar filtros, convertir espacios de color y mostrar resultados [3]. En este proyecto se usará desde Python por su facilidad de aprendizaje y rapidez de prototipado.

13.7.3. Espacios de color

Las cámaras suelen entregar imágenes en formato RGB o BGR. Sin embargo, para detección por color es común convertir la imagen al espacio HSV, que separa la información de tono, saturación y valor [4].

- **H:** tono o color principal.
- **S:** saturación o intensidad del color.
- **V:** brillo o valor.

Esta separación facilita detectar colores, aunque existan variaciones moderadas de iluminación.

13.7.4. Segmentación por umbral

La segmentación por color consiste en definir un rango mínimo y máximo para HSV. Los píxeles dentro del rango se consideran parte del objeto y los demás se descartan.

$$Borde = \begin{cases} 1, & \text{si } HSV_{min} \leq HSV(x, y) \leq HSV_{max} \\ 0, & \text{en otro caso} \end{cases}$$

13.7.5. Contornos y centroide

Después de obtener la máscara, se pueden identificar contornos. El contorno con mayor área suele corresponder al objeto principal. A partir de sus momentos geométricos se calcula el centroide [5]:

$$c_x = \frac{M_{10}}{M_{00}}$$
$$c_y = \frac{M_{01}}{M_{00}}$$

donde M_{00} , M_{10} y M_{01} son momentos de la región detectada.

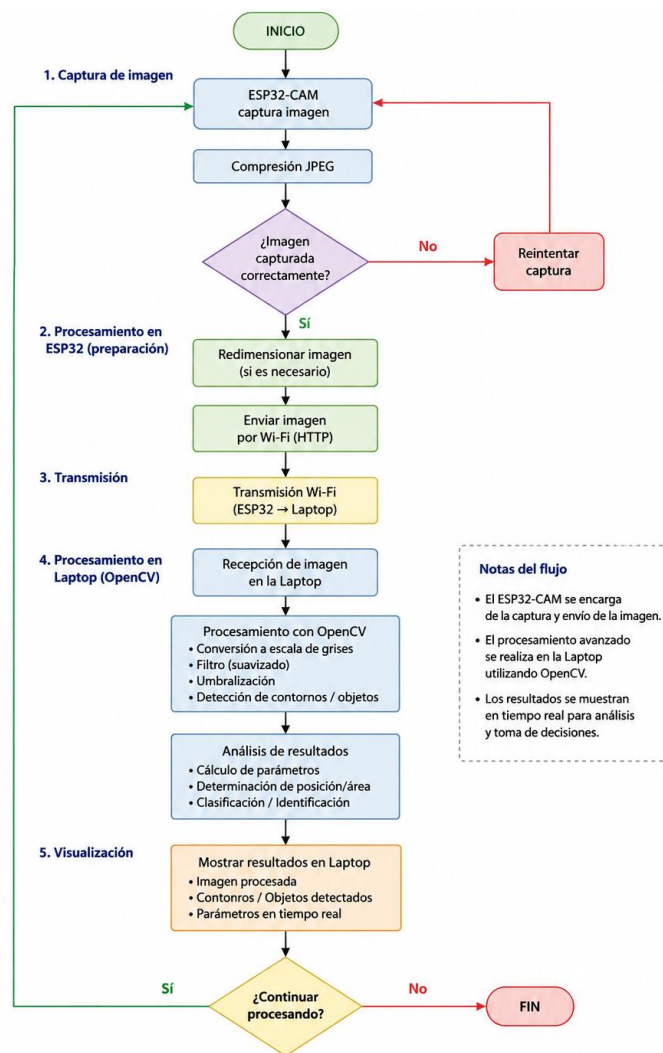


Figura 13.4: Flujo básico de procesamiento de imagen para detección por color.

13.8. Diseño electrónico

La ESP32-CAM requiere conexión a un programador USB-TTL para cargar el código. Durante la programación, el pin GPIO0 debe conectarse a GND para activar el modo de carga. Luego de programar, se desconecta GPIO0 de GND y se reinicia la placa.

Tabla 13.2: Conexión típica para programar una ESP32-CAM

USB-TTL	ESP32-CAM	Función
5 V	5 V	Alimentación
GND	GND	Tierra común
TX	U0R	Recepción de datos en ESP32-CAM
RX	U0T	Transmisión de datos desde ESP32-CAM
GND	GPIO0	Modo programación

La ESP32-CAM puede ser inestable si se alimenta desde una fuente débil. Para pruebas confiables se recomienda una fuente de 5 V con corriente suficiente y cables cortos.

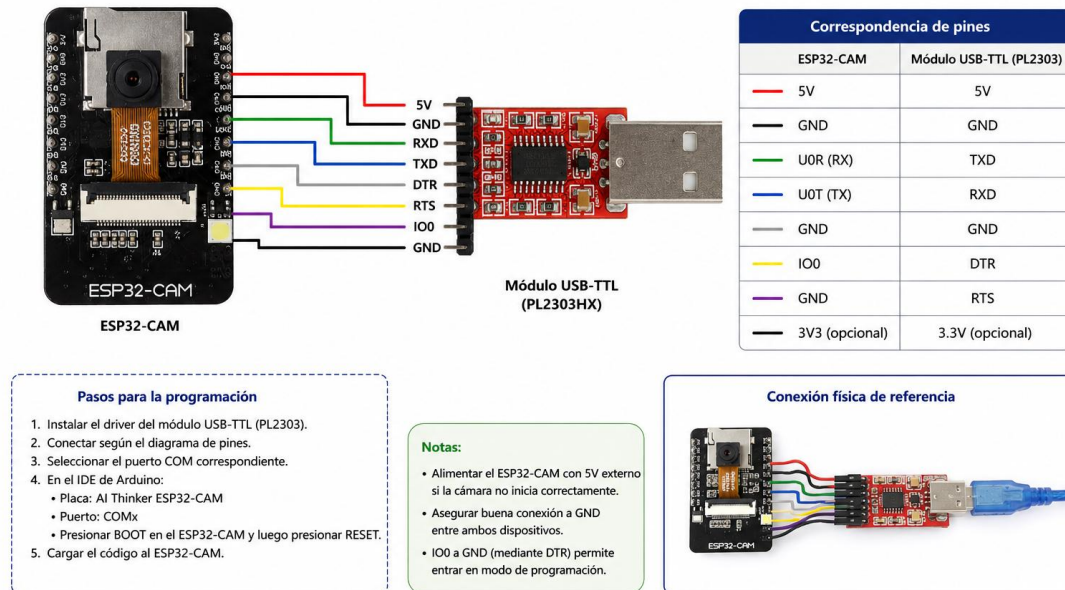


Figura 13.5: Conexión de programación de la ESP32-CAM.

13.9. Código para ESP32-CAM

Para la etapa de cámara se recomienda partir del ejemplo CameraWebServer disponible en Arduino IDE al instalar el soporte de ESP32. El siguiente fragmento resume la configuración principal para el modelo AI Thinker.

```
#include "esp_camera.h"
2 #include <WiFi.h>

const char* ssid = "NOMBRE_DE_RED";
const char* password = "CLAVE_DE_RED";

#define PWDN_GPIO_NUM 32
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM 0
#define SIOD_GPIO_NUM 26
#define SIOC_GPIO_NUM 27

#define Y9_GPIO_NUM 35
#define Y8_GPIO_NUM 34
#define Y7_GPIO_NUM 39
#define Y6_GPIO_NUM 36
```

```

#define Y5_GPIO_NUM 21
#define Y4_GPIO_NUM 19
#define Y3_GPIO_NUM 18
#define Y2_GPIO_NUM 5

#define VSYNC_GPIO_NUM 25
#define HREF_GPIO_NUM 23
#define PCLK_GPIO_NUM 22

void startCameraServer();

void setup()
{
    Serial.begin(115200);

    camera_config_t config;
    config.ledc_channel = LEDC_CHANNEL_0;
    config.ledc_timer = LEDC_TIMER_0;

    config.pin_d0 = Y2_GPIO_NUM;
    config.pin_d1 = Y3_GPIO_NUM;
    config.pin_d2 = Y4_GPIO_NUM;
    config.pin_d3 = Y5_GPIO_NUM;
    config.pin_d4 = Y6_GPIO_NUM;
    config.pin_d5 = Y7_GPIO_NUM;
    config.pin_d6 = Y8_GPIO_NUM;
    config.pin_d7 = Y9_GPIO_NUM;

    config.pin_xclk = XCLK_GPIO_NUM;
    config.pin_pclk = PCLK_GPIO_NUM;
    config.pin_vsync = VSYNC_GPIO_NUM;
    config.pin_href = HREF_GPIO_NUM;

    config.pin_sccb_sda = SIOD_GPIO_NUM;
    config.pin_sccb_scl = SIOC_GPIO_NUM;

    config.pin_pwdn = PWDN_GPIO_NUM;
    config.pin_reset = RESET_GPIO_NUM;

    config.xclk_freq_hz = 20000000;
    config.pixel_format = PIXFORMAT_JPEG;
    config.frame_size = FRAMESIZE_QVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;

    esp_err_t err = esp_camera_init(&config);

```

```

if (err != ESP_OK)
{
  Serial.printf("Error al iniciar camara: 0x %x", err);
  return;
}

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED)
{
  delay(500);
  Serial.print(".");
}

startCameraServer();

Serial.print("Camara lista en: http://");
Serial.println(WiFi.localIP());
}

void loop()
{
  delay(1000);
}

```

El ejemplo completo CameraWebServer incluye la función startCameraServer(). El fragmento anterior sirve para comprender la configuración, pero en laboratorio es conveniente usar el ejemplo completo del entorno Arduino-ESP32.

13.10. Prueba del video

Después de cargar el programa:

1. Retirar la conexión GPIO0-GND.
2. Reiniciar la ESP32-CAM.
3. Abrir el monitor serial.
4. Copiar la dirección IP mostrada.
5. Abrir la dirección en un navegador.
6. Verificar que se muestre la imagen de la cámara.

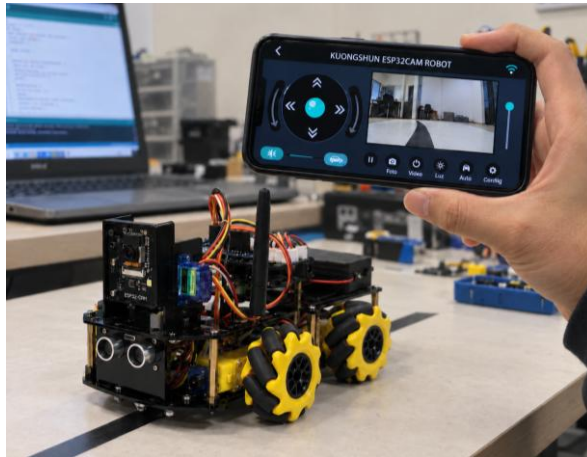


Figura 13.6: Prueba de transmisión de video desde la ESP32-CAM.

13.11. Código Python con OpenCV

El siguiente programa captura el video de la ESP32-CAM y detecta un objeto de color. El ejemplo está configurado para detectar un objeto rojo, pero los rangos HSV deben ajustarse según la iluminación y el color real [6].

```
import cv2
import numpy as np

# URL del stream de la ESP32-CAM
url = "http://192.168.1.100:81/stream"

cap = cv2.VideoCapture(url)

if not cap.isOpened():
    print("No se pudo abrir el video")
    exit()

while True:
    ret, frame = cap.read()

    if not ret:
        print("No se recibio imagen")
        break

    frame = cv2.resize(frame, (320, 240))
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    rojo_bajol = np.array([0, 100, 80])
    rojo_alto1 = np.array([10, 255, 255])
    rojo_bajo2 = np.array([170, 100, 80])
```

```

rojo_alto2 = np.array([180, 255, 255])

mascara1 = cv2.inRange(hsv, rojo_bajo1, rojo_alto1)
mascara2 = cv2.inRange(hsv, rojo_bajo2, rojo_alto2)
mascara = mascara1 + mascara2

mascara = cv2.erode(mascara, None, iterations=2)
mascara = cv2.dilate(mascara, None, iterations=2)

contornos, _ = cv2.findContours(
mascara,
cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE
)

comando = "SIN_OBJETO"

if len(contornos) > 0:
mayor = max(contornos, key=cv2.contourArea)
area = cv2.contourArea(mayor)

if area > 500:
M = cv2.moments(mayor)

5if M["m00"] != 0:
cx = int(M["m10"] / M["m00"])
cy = int(M["m01"] / M["m00"])

cv2.circle(frame, (cx, cy), 6, (0, 255, 0), -1)
cv2.drawContours(frame, [mayor], -1, (0, 255, 0), 2)

ancho = frame.shape[1]

if cx < ancho / 3:
comando = "IZQUIERDA"
elif cx > 2 * ancho / 3:
comando = "DERECHA"
else:
comando = "CENTRO"

cv2.putText(
frame,
comando,
(10, 30),
cv2.FONT_HERSHEY_SIMPLEX,
1,
(0, 255, 0),

```

```

2
)

cv2.imshow("Camara ESP32-CAM", frame)
cv2.imshow("Mascara", mascara)

print("Comando:", comando)

if cv2.waitKey(1) & 0xFF == ord("q"):
    break
cap.release()
cv2.destroyAllWindows()

```

13.12. Interpretación de la salida

El programa clasifica la posición del objeto en tres zonas:

- **IZQUIERDA:** el objeto está en el tercio izquierdo de la imagen.
- **CENTRO:** el objeto está en la zona central.
- **DERECHA:** el objeto está en el tercio derecho.
- **SIN_OBJETO:** no se detecta un objeto suficientemente grande.

Esta clasificación puede convertirse en comandos para un robot:

Tabla 13.3: Conversión de detección visual en comandos de movimiento.

Resultado visual	Comando sugerido	Acción del robot
IZQUIERDA	LEFT	Girar hacia la izquierda
CENTRO	FWD	Avanzar hacia el objeto
DERECHA	RIGHT	Girar hacia la derecha
SIN_OBJETO	SEARCH	Buscar objeto



Figura 13.7: Clasificación de posición horizontal del objeto detectado.

13.13. Ajuste de color HSV

La detección por color depende de la iluminación. Para ajustar los rangos HSV se recomienda utilizar una herramienta con barras deslizantes. El siguiente pseudocódigo resume la lógica de calibración:

```
Crear ventana de calibracion
Crear barras para H_min, S_min, V_min
Crear barras para H_max, S_max, V_max

Mientras el programa este activo:
Capturar imagen
Convertir imagen a HSV
Leer valores de barras
Crear mascara con rangos HSV
Mostrar imagen original y mascara
```

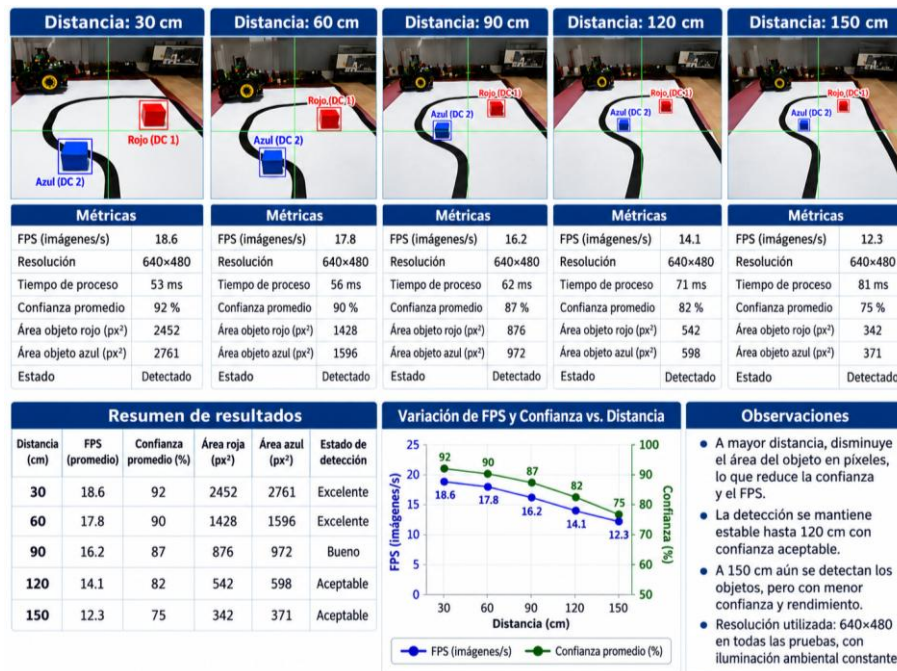


Figura 13.8: Calibración de rangos HSV para detección de color.

13.14. Integración con un robot móvil

Existen varias formas de integrar la detección visual con un robot [7]:

- Computadora procesa imagen y envía comandos por Bluetooth a un robot.
- Computadora procesa imagen y envía comandos por serial USB.
- ESP32-CAM transmite imagen y otro microcontrolador ejecuta movimiento.
- ESP32-CAM se utiliza solo para visualización y depuración.

En una primera versión educativa, se recomienda mantener separado el procesamiento de visión y el control de motores. La computadora analiza la imagen y envía comandos simples al robot.

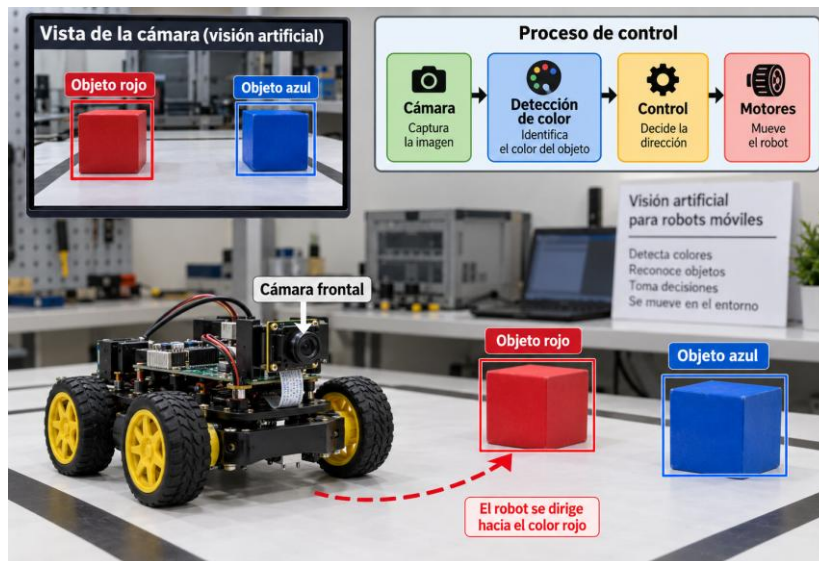


Figura 13.9: Robot móvil controlado mediante detección visual de color.

13.15. Limitaciones prácticas

La visión artificial con ESP32-CAM y OpenCV tiene limitaciones que deben considerarse:

- La ESP32-CAM no ofrece la misma calidad que una cámara industrial.
- La transmisión WiFi puede generar latencia.
- La iluminación afecta fuertemente la detección por color.
- Los reflejos pueden producir falsas detecciones.
- Los objetos pequeños pueden perderse a distancia.
- La detección por color no distingue objetos con colores similares.
- La velocidad del robot debe ser compatible con la latencia del sistema.

Para robots rápidos, la visión debe usarse con cuidado. En muchos casos conviene combinar cámara con sensores rápidos como distancia, reflectivos o encoders.

13.16. Pruebas experimentales

13.16.1. Prueba 1: transmisión de cámara

Verificar que la ESP32-CAM transmita imagen estable en el navegador.

13.16.2. Prueba 2: captura desde Python

Confirmar que OpenCV puede abrir el stream de video.

13.16.3. Prueba 3: detección de color

Colocar un objeto de color frente a la cámara y observar la máscara binaria.

13.16.4. Prueba 4: posición del objeto

Mover el objeto hacia izquierda, centro y derecha. Verificar que el programa genere el comando correcto.

13.16.5. Prueba 5: integración con robot

Enviar comandos hacia una plataforma móvil a baja velocidad y observar su respuesta.

Tabla 13.4: Registro sugerido de pruebas de visión artificial.

Prueba	Condición	Resultado esperado	Resultado obtenido
Stream	Cámara conectada	Video estable	
OpenCV	Captura desde Python	Imagen en ventana	
Color	Objeto rojo	Máscara correcta	
Posición	Objeto izquierda	Comando IZQUIERDA	
Integración	Robot móvil	Respuesta coherente	

13.17. Errores frecuentes

Durante el desarrollo pueden presentarse los siguientes errores:

- Alimentar la ESP32-CAM con una fuente insuficiente.
- Olvidar desconectar GPIO0 de GND después de programar.
- Seleccionar un modelo de cámara incorrecto en el código.
- Usar una URL de stream incorrecta.
- No instalar OpenCV en Python.
- Definir rangos HSV demasiado estrechos.
- Probar con iluminación muy variable.
- Colocar la cámara con enfoque u orientación inadecuados.

13.18. Proyecto de extensión

Como extensión, se propone enviar los comandos de visión desde Python hacia un ESP32 receptor mediante Bluetooth o WiFi. También puede implementarse una interfaz gráfica que muestre la imagen procesada, comandos generados, área del objeto y posición del centroide.

Otra mejora consiste en reemplazar la detección por color por detección de formas, códigos QR, marcadores ArUco o modelos de aprendizaje automático livianos, según el nivel del curso y los recursos disponibles.



Figura 13.10: Extensión del sistema de visión con interfaz de monitoreo.

Actividades de aprendizaje

1. Configure una ESP32-CAM y verifique la transmisión de video en un navegador.
2. Instale OpenCV en Python y capture el video desde la ESP32-CAM.
3. Ajuste los rangos HSV para detectar un objeto de color azul o verde.
4. Modifique el programa para mostrar el área del objeto detectado.
5. Clasifique la posición del objeto en cinco zonas en lugar de tres.
6. Proponga una integración con un robot móvil.

Preguntas de reflexión

1. ¿Qué ventajas ofrece una cámara frente a sensores de distancia simples?
2. ¿Por qué el espacio HSV es útil para detección por color?
3. ¿Cómo afecta la iluminación al desempeño del sistema?
4. ¿Qué problemas genera la latencia en un robot móvil rápido?
5. ¿Por qué conviene iniciar con comandos simples como izquierda, centro y derecha?
6. ¿Cómo podría combinar visión artificial con sensores reflectivos o ultrasónicos?

Referencias

[1] Espressif Systems, “ESP32 Camera Driver”, Espressif GitHub Documentation. [En línea]. Disponible en: <https://github.com/espressif/esp32-camera>. [Accedido: 24- jun-2026].

- [2] Espressif Systems, “Wi-Fi API”, Arduino ESP32 Documentation. [En línea]. Disponible en: <https://docs.espressif.com/projects/arduino-esp32/en/latest/api/wifi.html>. [Accedido: 24-jun-2026].
- [3] OpenCV, “OpenCV Documentation”, OpenCV. [En línea]. Disponible en: <https://docs.opencv.org/>. [Accedido: 24-jun-2026].
- [4] OpenCV, “Changing Colorspaces”, OpenCV-Python Tutorials. [En línea]. Disponible en: https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html. [Accedido: 24-jun-2026].
- [5] OpenCV, “Contours in OpenCV”, OpenCV-Python Tutorials. [En línea]. Disponible en: https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html. [Accedido: 24-jun-2026].
- [6] Python Software Foundation, “Python Documentation”. [En línea]. Disponible en: <https://docs.python.org/3/>. [Accedido: 24-jun-2026].
- [7] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.
- [8] R. Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed. Cham, Switzerland: Springer, 2022.

Capítulo 14. Robot Explorador WiFi con ESP32-CAM

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender la arquitectura de un robot explorador controlado por WiFi.
- Integrar una ESP32-CAM como módulo de video en tiempo real.
- Diseñar una interfaz web básica para controlar movimiento desde un navegador.
- Separar funciones de visión, control de motores y alimentación para mejorar la confiabilidad del robot.
- Implementar comunicación serial entre una ESP32-CAM y un microcontrolador auxiliar.
- Controlar motores DC mediante comandos de texto enviados desde una interfaz web.
- Identificar problemas prácticos de latencia, alimentación, alcance WiFi, cableado y estabilidad mecánica.

14.1. Introducción

Un robot explorador es una plataforma móvil diseñada para desplazarse en un entorno y enviar información al operador. Puede utilizarse en educación, inspección básica, monitoreo remoto, demostraciones tecnológicas, ferias de robótica o proyectos de investigación inicial. A diferencia de un robot completamente autónomo, un robot explorador puede ser controlado por una persona mediante una interfaz remota, mientras transmite video o datos del entorno [6].

La ESP32-CAM permite construir robots exploradores de bajo costo, ya que integra cámara y conectividad WiFi en una placa compacta. Sin embargo, usar la ESP32-CAM como único controlador para video y motores puede generar limitaciones: pocos pines disponibles, alimentación sensible, reinicios por consumo, interferencia entre motores y cámara, y dificultad para depurar. Por ello, en este capítulo se propone una arquitectura modular: la ESP32-CAM se encarga del video y de la interfaz web, mientras que un microcontrolador auxiliar, como Arduino Nano, controla los motores [1].

Esta separación permite que el robot sea más confiable y fácil de modificar. La ESP32-CAM recibe órdenes desde el navegador y las envía por comunicación serial al Arduino Nano. El Arduino interpreta comandos como avanzar, retroceder, girar o detenerse, y controla el driver de motores.

El proyecto continúa el trabajo del Capítulo 13, donde se introdujo la ESP32-CAM y OpenCV. En este capítulo, la cámara ya no se utiliza solo para procesamiento externo, sino como parte de un robot móvil controlado por una interfaz web.

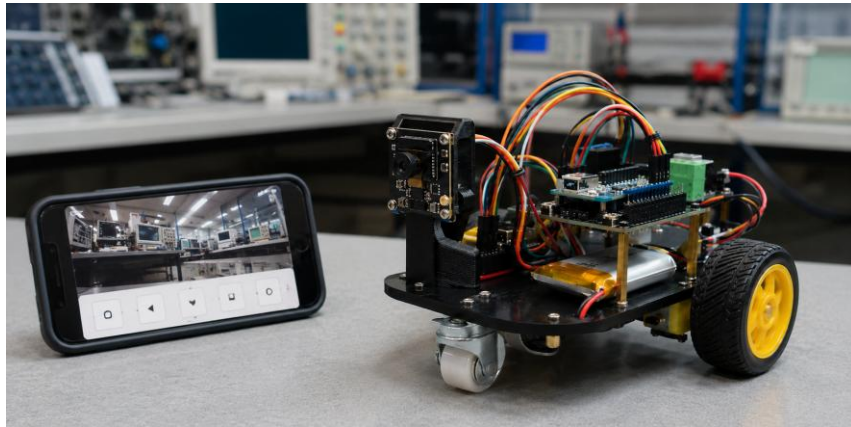


Figura 14.1: Robot explorador WiFi con cámara y control desde navegador.

14.2. Planteamiento del problema

Construir un robot móvil con cámara parece sencillo: se instala una cámara, se conectan motores y se controla por WiFi. En la práctica, aparecen varios problemas:

- La ESP32-CAM puede reiniciarse si la fuente no entrega corriente suficiente.
- Los motores generan ruido eléctrico que afecta la cámara o la comunicación.
- La placa tiene pocos pines disponibles para sensores y actuadores.
- La transmisión de video consume recursos y puede aumentar la latencia.
- El robot puede seguir moviéndose si se pierde comunicación.
- Una interfaz web mal diseñada puede ser lenta o incómoda.
- El cableado puede volverse difícil de mantener si no se organiza por módulos.

La solución propuesta consiste en dividir el sistema en bloques funcionales y establecer una comunicación simple entre ellos.

Un robot explorador confiable no depende solo de tener cámara. Requiere una arquitectura clara: video, control, potencia, motores y seguridad deben estar organizados como subsistemas.

14.3. Descripción general del proyecto

El robot explorador propuesto incluye:

- **ESP32-CAM:** transmite video y aloja una interfaz web.
- **Arduino Nano:** recibe comandos por serial y controla motores.
- **Driver TB6612FNG:** controla dos motores DC.
- **Batería:** alimenta motores y electrónica mediante regulación adecuada.
- **Chasis móvil:** estructura mecánica con dos ruedas motrices y rueda loca.
- **Interfaz web:** botones para avanzar, retroceder, girar y detener.

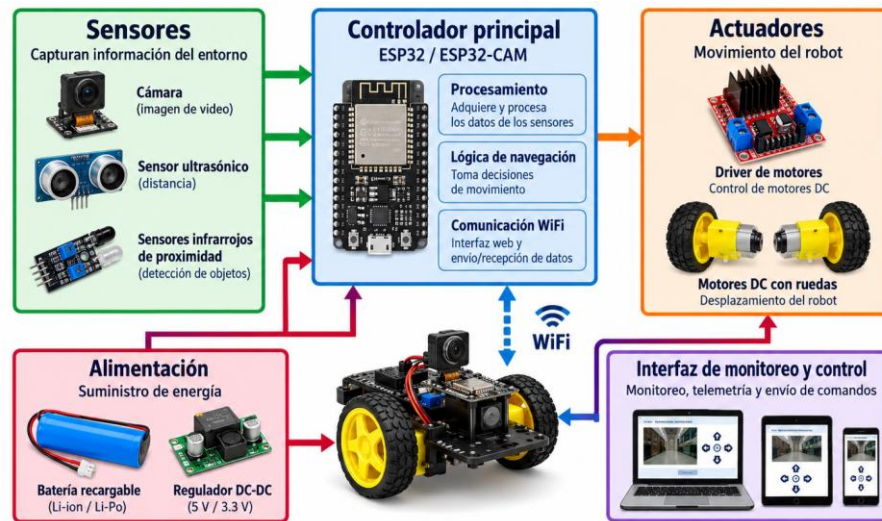


Figura 14.2: Arquitectura general del robot explorador WiFi.

14.4. Aplicación STEM

Este proyecto integra las áreas STEM de la siguiente forma:

- **Ciencia:** análisis de movimiento, energía, transmisión inalámbrica y percepción visual.
- **Tecnología:** uso de ESP32-CAM, WiFi, interfaz web, microcontroladores y drivers de motor.
- **Ingeniería:** diseño modular de un robot móvil con cámara y control remoto.
- **Matemáticas:** control diferencial, velocidades PWM, tiempos de seguridad y análisis de consumo.

14.5. Objetivos del proyecto

14.5.1. Objetivo general

Diseñar e implementar un robot explorador WiFi con ESP32-CAM, capaz de transmitir video en tiempo real y recibir comandos de movimiento desde una interfaz web local.

14.5.2. Objetivos específicos

1. Configurar la ESP32-CAM como punto de acceso WiFi o como estación conectada a una red local.
2. Crear una interfaz web con botones de control.
3. Enviar comandos desde la ESP32-CAM hacia un Arduino Nano mediante comunicación serial.
4. Controlar motores DC mediante un driver TB6612FNG.

5. Implementar una parada automática si no se reciben comandos durante un tiempo definido.
6. Evaluar el comportamiento del robot en pruebas de movimiento y transmisión de video.

14.6. Materiales y herramientas

Tabla 14.1: Materiales principales para el robot explorador WiFi.

Elemento	Cantidad	Función
ESP32-CAM AI Thinker o similar	1	Video e interfaz web
Arduino Nano	1	Control de motores
Driver TB6612FNG	1	Control de dos motores DC
Motores DC con caja reductora	2	Tracción diferencial
Chasis móvil	1	Estructura del robot
Ruedas	2	Movimiento
Rueda loca o caster	1	Apoyo frontal o posterior
Batería LiPo o power bank	1	Alimentación
Regulador DC-DC	1	Alimentación estable de electrónica
Módulo USB-TTL	1	Programación de ESP32-CAM
Celular o computadora	1	Control desde navegador

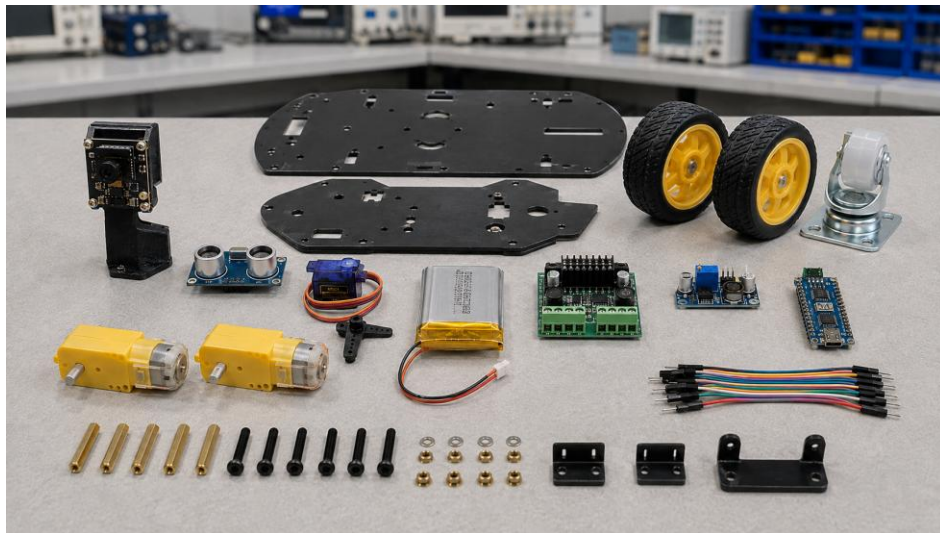


Figura 14.3: Materiales utilizados para construir el robot explorador WiFi.

14.7. Fundamento teórico

14.7.1. Control remoto por WiFi

El control remoto por WiFi permite enviar comandos desde un navegador hacia el robot. La ESP32-CAM puede operar como estación conectada a un router o como punto de acceso. En

modo punto de acceso, el robot crea su propia red WiFi, lo cual resulta útil en laboratorios o demostraciones donde no se desea depender de internet [2].

14.7.2. Arquitectura con dos microcontroladores

Aunque la ESP32-CAM puede controlar algunos actuadores, una arquitectura con dos microcontroladores ofrece ventajas [7]:

- Se reduce la carga de procesamiento en la ESP32-CAM.
- El Arduino Nano controla motores de forma más simple y estable.
- Se facilita depuración de video y movimiento por separado.
- Se mejora la modularidad del proyecto.
- Se pueden reutilizar códigos de control de motores desarrollados en capítulos previos.

14.7.3. Comunicación serial entre módulos

La comunicación serial permite enviar comandos de texto desde la ESP32-CAM hacia el Arduino Nano. Por ejemplo [4]:

- **F:** avanzar.
- **B:** retroceder.
- **L:** girar izquierda.
- **R:** girar derecha.
- **S:** detener.

Esta estrategia es simple, legible y fácil de depurar desde el monitor serial.

14.7.4. Control diferencial

Un robot con dos motores laterales se controla mediante velocidades independientes:

$$V_{izq} = PWM_{izq}$$

$$V_{der} = PWM_{der}$$

Para avanzar, ambos motores giran hacia adelante. Para girar, los motores giran en sentidos opuestos o con velocidades diferentes. Este principio se utilizó en capítulos anteriores y se mantiene en este proyecto.

14.8. Diseño electrónico

La ESP32-CAM y el Arduino Nano deben compartir tierra común para que la comunicación serial sea confiable. Como la ESP32-CAM trabaja a 3.3 V y el Arduino Nano clásico trabaja a 5 V, se recomienda cuidar los niveles lógicos. La línea TX de ESP32-CAM hacia RX de Arduino suele ser aceptada por el Arduino como nivel alto. En sentido inverso, si se envían datos desde Arduino hacia ESP32-CAM, se recomienda usar divisor de voltaje o convertor de nivel.

Tabla 14.2: Conexiones principales entre ESP32-CAM y Arduino Nano.

ESP32-CAM	Arduino Nano	Función
GND	GND	Tierra común
GPIO 14 / TX auxiliar	D 2 / RX SoftwareSerial	Comandos hacia Arduino
5 V regulados	5V o VIN según diseño	Alimentación adecuada

No se debe conectar una señal de 5 V directamente a un pin de entrada de la ESP32-CAM. Si el Arduino envía datos de regreso hacia la ESP32-CAM, se debe usar adaptación de nivel.

Tabla 14.3: Conexiones sugeridas entre Arduino Nano y TB6612FNG.

Arduino Nano	TB6612FNG	Función
D3	PWMA	PWM motor izquierdo
D4	AIN1	Dirección motor izquierdo
D5	AIN2	Dirección motor izquierdo
D6	PWMB	PWM motor derecho
D7	BIN1	Dirección motor derecho
D8	BIN2	Dirección motor derecho
D9	STBY	Habilitación del driver
GND	GND	Tierra común

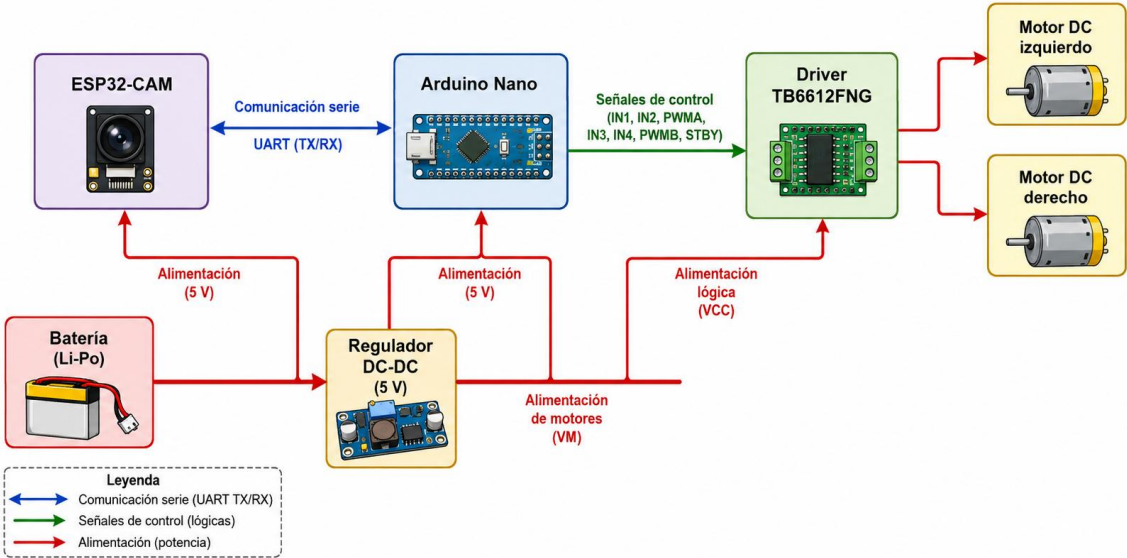


Figura 14.4: Diagrama de conexiones del robot explorador WiFi.

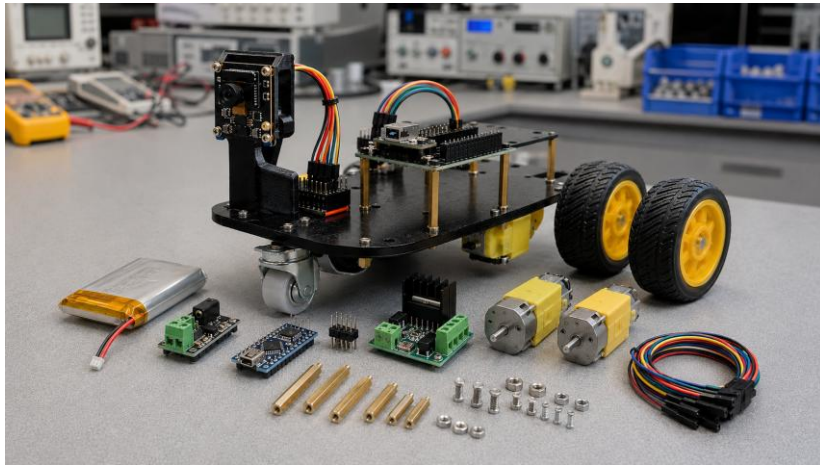


Figura 14.5: Montaje electrónico y mecánico del robot explorador.

14.9. Diseño de la interfaz web

La interfaz debe incluir video y botones grandes para control. En una pantalla de celular, los botones deben ser fáciles de presionar [3].

Elementos recomendados:

- Visualización de video.
- Botón avanzar.
- Botón retroceder.
- Botón izquierda.
- Botón derecha.
- Botón detener.
- Indicador de comando actual.

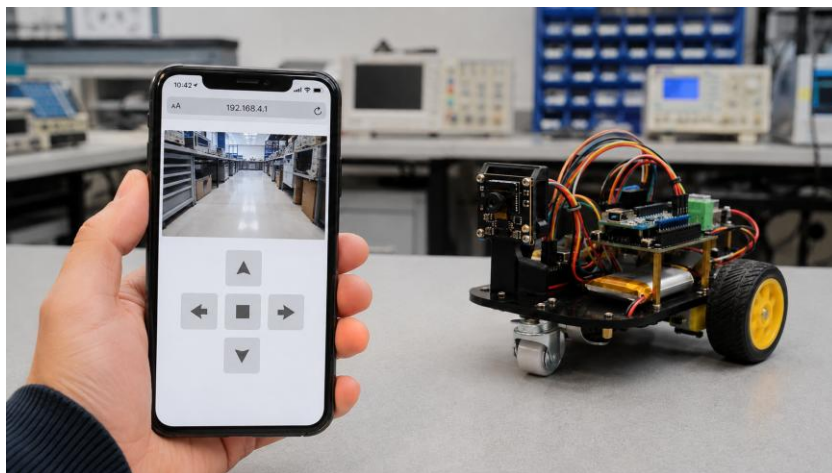


Figura 14.6: Interfaz web sugerida para control del robot explorador.

14.10. Código para Arduino Nano: control de motores

El siguiente programa recibe comandos por SoftwareSerial y controla los motores mediante TB6612FNG. Además, incorpora una parada automática si no se recibe ningún comando durante un tiempo determinado [5].

```
#include <SoftwareSerial.h>

// =====
// COMUNICACION CON ESP32-CAM
// =====
const byte RX_CMD = 2; // recibe desde TX de ESP32-CAM
const byte TX_CMD = 10; // opcional, no usado inicialmente

SoftwareSerial enlace(RX_CMD, TX_CMD);

// =====
// DRIVER TB6612FNG
// =====
const byte PWMA = 3;
const byte AIN1 = 4;
const byte AIN2 = 5;

const byte PWMB = 6;
const byte BIN1 = 7;
const byte BIN2 = 8;

const byte STBY = 9;

// =====
// PARAMETROS
// =====
int velocidad = 160;
const unsigned long TIEMPO_SEGURIDAD = 800;
unsigned long ultimoComando = 0;

// =====
// MOTORES
// =====
void motorIzquierdo(int v)
{
    v = constrain(v, -255, 255);

    if (v > 0)
    {
        digitalWrite(AIN1, HIGH);
        digitalWrite(AIN2, LOW);
    }
}
```

```

    analogWrite(PWMA, v);
}
else if (v < 0)
{
    digitalWrite(AIN1, LOW);
    digitalWrite(AIN2, HIGH);
    analogWrite(PWMA, -v);
}
else
{
    digitalWrite(AIN1, LOW);
    digitalWrite(AIN2, LOW);
    analogWrite(PWMA, 0);
}
}

void motorDerecho(int v)
{
    v = constrain(v, -255, 255);

    if (v > 0)
    {
        digitalWrite(BIN1, HIGH);
        digitalWrite(BIN2, LOW);
        analogWrite(PWMB, v);
    }
    else if (v < 0)
    {
        digitalWrite(BIN1, LOW);
        digitalWrite(BIN2, HIGH);
        analogWrite(PWMB, -v);
    }
    else
    {
        digitalWrite(BIN1, LOW);
        digitalWrite(BIN2, LOW);
        analogWrite(PWMB, 0);
    }
}

void mover(int izq, int der)
{
    motorIzquierdo(izq);
    motorDerecho(der);
}

void detener()

```

```

{
  mover(0, 0);
}

// =====
// COMANDOS
// =====
void ejecutarComando(char c)
{
  ultimoComando = millis();

  switch (c)
  {
    case 'F':
      mover(velocidad, velocidad);
      break;

    case 'B':
      mover(-velocidad, -velocidad);
      break;

    case 'L':
      mover(-velocidad, velocidad);
      break;

    case 'R':
      mover(velocidad, -velocidad);
      break;

    case 'S':
      detener();
      break;

    case '+':
      velocidad += 10;
      velocidad = constrain(velocidad, 0, 255);
      break;

    case '-':
      velocidad -= 10;
      velocidad = constrain(velocidad, 0, 255);
      break;
  }
}

// =====
// SETUP Y LOOP

```

```
// =====
void setup()
{
  pinMode(PWMA, OUTPUT);
  pinMode(AIN1, OUTPUT);
  pinMode(AIN2, OUTPUT);

  pinMode(PWMB, OUTPUT);
  pinMode(BIN1, OUTPUT);
  pinMode(BIN2, OUTPUT);

  pinMode(STBY, OUTPUT);
  digitalWrite(STBY, HIGH);

  Serial.begin(115200);
  enlace.begin(9600);

  detener();

  Serial.println("Control de robot explorador iniciado");
}

void loop()
{
  if (enlace.available())
  {
    char c = enlace.read();
    ejecutarComando(c);

    Serial.print("Comando recibido: ");
    Serial.println(c);
  }

  if (millis() - ultimoComando > TIEMPO_SEGURIDAD)
  {
    detener();
  }
}
```

14.11. Código para ESP32-CAM: interfaz de comandos

La implementación completa de video se recomienda realizar a partir del ejemplo *CameraWebServer*. Para añadir control, se agregan rutas HTTP que envían caracteres por serial al Arduino Nano. El siguiente fragmento muestra la lógica de comandos:

```
#include "esp_camera.h"
#include <WiFi.h>
```

```

#include <WebServer.h>

const char* ssid = "EXPLORADOR_WIFI";
const char* password = "12345678";

WebServer server(80);

// Usar un puerto serial auxiliar si la placa y pines lo permiten
HardwareSerial MotorSerial(1);

void enviarComando(char c)
{
    MotorSerial.write(c);
}

String paginaControl()
{
    String html = "";
    html += "<!DOCTYPE html><html><head>";
    html += "<meta name='viewport' content='width=device-width, initial-scale=1'>";
    html += "<title>Robot Explorador</title>";
    html += "<style>";
    html += "body{font-family:Arial;text-align:center;background:#111;color:white;}";
    html += "button{width:100px;height:70px;font-size:26px;margin:8px;border-radius:12px;}";
    html += ".stop{background:#b00020;color:white;}";
    html += "</style></head><body>";
    html += "<h2>Robot Explorador WiFi</h2>";
    html += "<p>Video: abrir stream de la ESP32-CAM</p>";
    html += "<div>";
    html += "<button onclick=\"cmd('F')\">↑</button><br>";
    html += "<button onclick=\"cmd('L')\">←</button>";
    html += "<button class='stop' onclick=\"cmd('S')\">STOP</button>";
    html += "<button onclick=\"cmd('R')\">→</button><br>";
    html += "<button onclick=\"cmd('B')\">↓</button>";
    html += "</div>";
    html += "<p>";
    html += "<button onclick=\"cmd('+')\">Vel +</button>";
    html += "<button onclick=\"cmd('-')\">Vel -</button>";
    html += "</p>";
    html += "<script>";
    html += "function cmd(c){fetch('/cmd?c='+c);}";
    html += "</script></body></html>";
    return html;
}

```

```

}

void manejarRaiz()
{
  server.send(200, "text/html", paginaControl());
}

void manejarComando()
{
  if (server.hasArg("c"))
  {
    char c = server.arg("c")[0];
    enviarComando(c);
    server.send(200, "text/plain", "OK");
  }
  else
  {
    server.send(400, "text/plain", "Sin comando");
  }
}

void setup()
{
  Serial.begin(115200);

  // Ejemplo: TX en GPIO14 hacia RX del Arduino Nano
  MotorSerial.begin(9600, SERIAL_8N1, -1, 14);

  WiFi.softAP(ssid, password);

  server.on("/", manejarRaiz);
  server.on("/cmd", manejarComando);
  server.begin();

  Serial.print("Red creada. IP: ");
  Serial.println(WiFi.softAPIP());

  // En una implementacion completa, aqui tambien se inicia la camara
  // usando el ejemplo CameraWebServer.
}

void loop()
{
  server.handleClient();
}

```


La ESP32-CAM tiene pines limitados y algunos se usan internamente para cámara o memoria microSD. Antes de elegir el pin serial hacia Arduino, se debe verificar disponibilidad según el modelo de placa.

14.12. Integración del video

El ejemplo anterior muestra la interfaz de comandos. Para una versión completa, se recomienda partir del ejemplo *CameraWebServer* y añadir las rutas /cmd y la página de control. Otra opción es usar dos páginas: una para video y otra para control.

En una práctica educativa, se puede iniciar de forma progresiva:

1. Probar primero video de la ESP32-CAM.
2. Probar después comunicación ESP32-CAM hacia Arduino.
3. Probar motores con comandos seriales.
4. Integrar interfaz web.
5. Unir video y control.

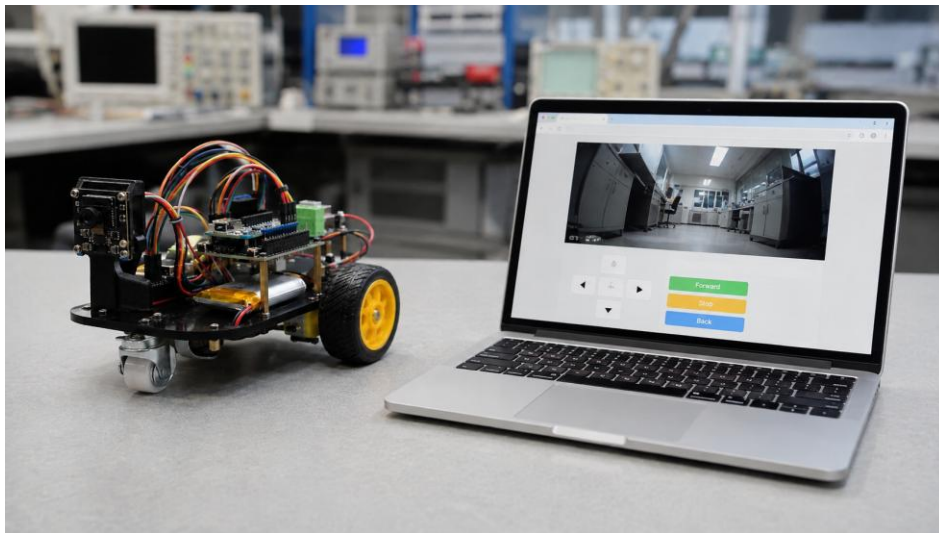


Figura 14.7: Integración de video y control web del robot explorador.

14.13. Pruebas experimentales

14.13.1. Prueba 1: control de motores sin cámara

Enviar comandos desde el monitor serial o desde la ESP32-CAM sin video activo. Verificar que el robot avance, retroceda, gire y se detenga.

14.13.2. Prueba 2: video estable

Activar la transmisión de video y verificar que no existan reinicios. Si la placa se reinicia, revisar alimentación.

14.13.3. Prueba 3: interfaz web

Probar cada botón desde el celular y observar la respuesta del robot.

14.13.4. Prueba 4: seguridad por pérdida de comandos

Dejar de presionar botones o desconectar el celular. Verificar que el Arduino detenga el robot después del tiempo de seguridad.

14.13.5. Prueba 5: prueba de exploración

Conducir el robot a baja velocidad en un entorno controlado, evitando obstáculos peligrosos, escaleras o superficies inestables.

Tabla 14.4: Registro sugerido de pruebas del robot explorador.

Prueba	Condición	Resultado esperado	Resultado obtenido
Motores	Comando F	Avanza	
Motores	Comando S	Se detiene	
Video	Cámara activa	Imagen estable	
Web	Botones desde celular	Respuesta correcta	
Seguridad	Sin comandos	Parada automática	

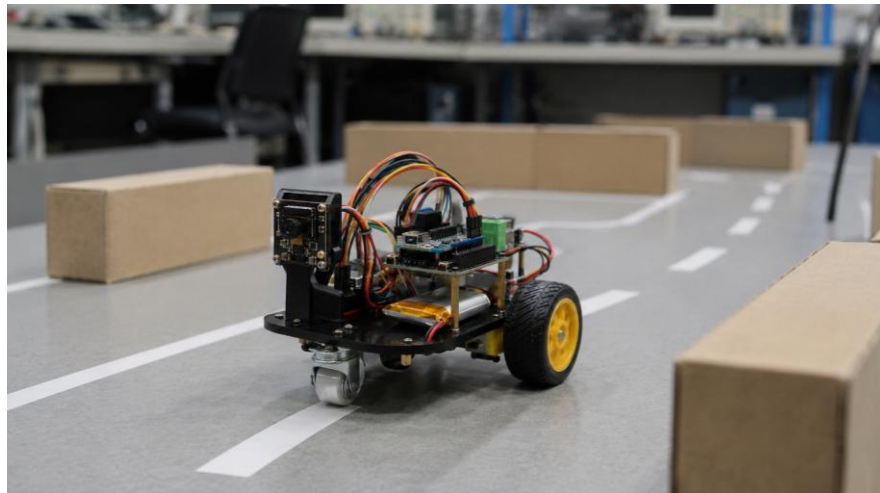


Figura 14.8: Prueba experimental del robot explorador WiFi.

14.14. Consideraciones de alimentación

La alimentación es uno de los puntos más importantes. Los motores pueden generar caídas de voltaje y ruido eléctrico. La ESP32-CAM requiere una fuente estable para evitar reinicios.

Recomendaciones:

- Usar un regulador independiente para electrónica.
- Separar alimentación de motores y lógica cuando sea posible.

- Mantener tierra común entre todos los módulos.
- Colocar capacitores cerca del driver y de la ESP32-CAM.
- Usar cables cortos y de calibre adecuado.
- Evitar alimentar motores desde el mismo pin de 5 V de la placa.

14.15. Problemas frecuentes

Durante el desarrollo pueden presentarse los siguientes errores:

- La ESP32-CAM se reinicia al mover motores.
- El video se congela por señal WiFi débil.
- El Arduino no recibe comandos por falta de tierra común.
- Se conectan niveles de 5 V hacia la ESP32-CAM sin adaptación.
- El robot sigue avanzando al perder conexión si no existe tiempo de seguridad.
- Los motores giran en sentido opuesto por cableado invertido.
- Los botones web envían comandos, pero el robot no responde por pin serial incorrecto.

14.16. Proyecto de extensión

Como extensión, se propone agregar sensores ultrasónicos o infrarrojos para evitar choques. El Arduino podría detener el robot automáticamente si detecta un obstáculo cercano, aunque el operador presione avanzar.

Otra extensión consiste en combinar este robot con el dashboard del Capítulo 12. Así, además de video y control, el usuario podría visualizar voltaje, corriente, temperatura y estado del robot.

También puede integrarse con el sistema de visión del Capítulo 13. En ese caso, la computadora procesaría la imagen y enviaría comandos automáticos al robot, permitiendo un modo semiautónomo.



Figura 14.9: Extensiones posibles del robot explorador WiFi.

Actividades de aprendizaje

1. Pruebe el código del Arduino Nano enviando comandos desde el monitor serial.
2. Configure la ESP32-CAM como punto de acceso WiFi.
3. Cree una interfaz web con botones de dirección.
4. Integre la comunicación serial entre ESP32-CAM y Arduino Nano.
5. Implemente una parada automática por pérdida de comandos.
6. Agregue un sensor de obstáculo y detención preventiva.

Preguntas de reflexión

1. ¿Por qué conviene separar cámara y control de motores en dos microcontroladores?
2. ¿Qué problemas puede causar una alimentación inestable en la ESP32-CAM?
3. ¿Por qué es importante implementar parada automática por pérdida de comandos?
4. ¿Qué ventajas ofrece una interfaz web frente a una aplicación móvil dedicada?
5. ¿Cómo integraría este robot con el dashboard de telemetría del Capítulo 13?
6. ¿Qué mejoras serían necesarias para convertirlo en un robot semiautónomo?

Referencias

- [1] Espressif Systems, “ESP32 Camera Driver”, Espressif GitHub Documentation. [En línea]. Disponible en: <https://github.com/espressif/esp32-camera>, [Accedido: 24- jun-2026].
- [2] Espressif Systems, “Wi-Fi API”, Arduino ESP32 Documentation. [En línea]. Disponible en: <https://docs.espressif.com/projects/arduino-esp32/en/latest/api/wifi.html>. [Accedido: 24-jun-2026].
- [3] Espressif Systems, “WebServer library for Arduino ESP32”, Arduino ESP32 Documentation. [En línea]. Disponible en: <https://docs.espressif.com/projects/arduino-esp32/en/latest/api/webserver.html>. [Accedido: 24-jun-2026].
- [4] Arduino, “SoftwareSerial Library”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/learn/built-in-libraries/software-serial>. [Accedido: 24-jun-2026].
- [5] Toshiba Electronic Devices & Storage Corporation, “TB6612FNG Brush Motor Driver IC”, Toshiba. [En línea]. Disponible en: <https://toshiba.semicon-storage.com/ap-en/semiconductor/product/motor-driver-ics/brushed-dc-motor-driver-ics/detail.TB6612FNG.html>. [Accedido: 24-jun-2026].
- [6] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.
- [7] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2017.

Capítulo 15. Odometría y Control de Velocidad con Encoders

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender la función de los encoders en robots móviles diferenciales.
- Medir pulsos de encoder mediante interrupciones externas.
- Calcular revoluciones por minuto, velocidad lineal y distancia recorrida.
- Estimar posición aproximada mediante odometría diferencial.
- Implementar control de velocidad por lazo cerrado en motores DC.
- Analizar errores de odometría producidos por deslizamiento, tolerancias mecánicas y diferencias entre motores.
- Integrar odometría básica en robots exploradores, seguidores de línea y plataformas educativas.

15.1. Introducción

Los robots móviles desarrollados en capítulos anteriores se han basado principalmente en sensores de entorno, como ultrasonido, cámara, sensores reflectivos y telemetría. Estos sensores permiten percibir obstáculos, líneas, colores o variables eléctricas. Sin embargo, para mejorar la navegación es necesario conocer también cómo se está moviendo el robot: cuánto avanzó, qué tan rápido giran sus ruedas y cuánto ha cambiado su orientación [5].

Los encoders permiten medir el giro de un eje o una rueda. Al contar pulsos generados por el encoder, el microcontrolador puede estimar revoluciones, velocidad, distancia recorrida y, en robots diferenciales, posición aproximada. Esta información es esencial para mejorar el control de movimiento, corregir diferencias entre motores y ejecutar trayectorias más repetibles.

En un robot explorador, la odometría permite avanzar una distancia específica o girar un ángulo aproximado. En un seguidor de línea, permite comparar velocidades reales de las ruedas y detectar deslizamiento. En un robot Minisumo o de combate, puede servir para registrar aceleraciones, analizar empuje o controlar maniobras temporizadas con mayor precisión [4].

Este capítulo desarrolla un sistema de odometría básica y control de velocidad con Arduino Nano, encoders de rueda y driver TB6612FNG. El proyecto se plantea como una evolución del robot explorador autónomo, permitiendo que el robot ya no dependa únicamente de tiempos fijos para avanzar o girar [3].

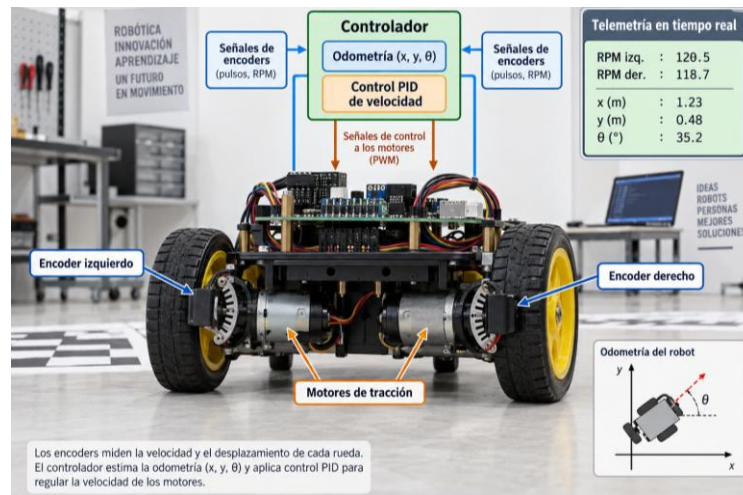


Figura 15.1: Robot móvil con encoders para odometría y control de velocidad.

15.2. Planteamiento del problema

En muchos robots educativos, las maniobras se programan usando tiempos:

```
mover(150, 150);
delay(1000);
detener();
```

Este método es simple, pero poco preciso. El robot puede avanzar una distancia diferente si cambia el voltaje de la batería, la fricción de la pista, el peso del robot o la carga de los motores. De igual manera, un giro de 500 ms no siempre produce el mismo ángulo.

Algunos problemas frecuentes del control por tiempo son:

- El robot avanza menos cuando la batería está baja.
- Un motor gira más rápido que el otro.
- Las ruedas patinan y la distancia real cambia.
- El robot no gira siempre el mismo ángulo.
- La velocidad no se mantiene constante ante cambios de carga.
- Las pruebas son difíciles de repetir.

Los encoders permiten cerrar el ciclo de control. En lugar de ordenar solo un PWM, el sistema mide lo que realmente ocurre en la rueda y ajusta la salida.

El control por tiempo ordena al robot moverse; la odometría permite medir cuánto se movió realmente.

15.3. Descripción general del proyecto

El sistema propuesto utiliza dos encoders, uno en cada rueda motriz. Cada encoder genera pulsos al girar.

El Arduino Nano cuenta esos pulsos mediante interrupciones y calcula:

- Pulsos por intervalo de tiempo.
- Revoluciones por minuto de cada rueda.
- Distancia recorrida por cada rueda.
- Velocidad lineal aproximada.
- Posición x , y y orientación θ estimadas.

Además, se implementa un controlador proporcional simple para mantener la velocidad deseada de cada motor.

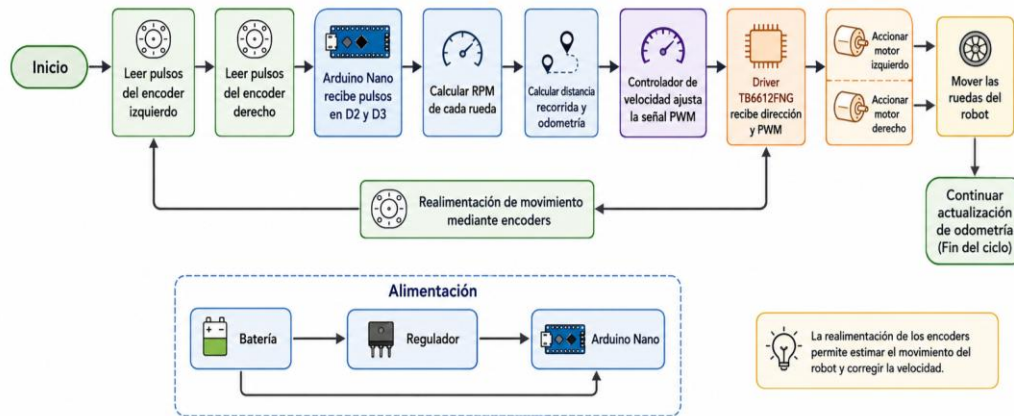


Figura 15.2: Arquitectura general del sistema de odometría y control de velocidad.

15.4. Aplicación STEM

Este proyecto integra las áreas STEM de la siguiente manera:

- **Ciencia:** análisis de movimiento, velocidad, distancia, fricción y deslizamiento.
- **Tecnología:** uso de encoders, interrupciones, microcontroladores y drivers de motor.
- **Ingeniería:** diseño de control de movimiento para robots móviles.
- **Matemáticas:** conversión de pulsos a distancia, cinemática diferencial y estimación de posición.

15.5. Objetivos del proyecto

15.5.1. Objetivo general

Diseñar e implementar un sistema de odometría básica y control de velocidad con encoders para robots móviles diferenciales.

15.5.2. Objetivos específicos

1. Leer pulsos de encoders usando interrupciones externas.
2. Calcular RPM y distancia recorrida de cada rueda.

3. Estimar desplazamiento y orientación del robot.
4. Implementar control de velocidad por lazo cerrado.
5. Registrar datos por monitor serial para análisis experimental.
6. Evaluar el efecto de batería, fricción y deslizamiento sobre la odometría.

15.6. Materiales y herramientas

Tabla 15.1: Materiales principales para el sistema de odometría.

Elemento	Cantidad	Función
Arduino Nano	1	Control y lectura de encoders
Motores DC con encoder	2	Tracción y medición de giro
Driver TB6612FNG	1	Control de motores DC
Ruedas	2	Movimiento del robot
Chasis diferencial	1	Plataforma móvil
Batería LiPo o pack recargable	1	Alimentación
Regulador DC-DC	1	Alimentación estable del control
Cables y conectores	Varios	Interconexión
Computadora con Arduino IDE	1	Programación y monitoreo
Cinta métrica o regla	1	Validación de distancia

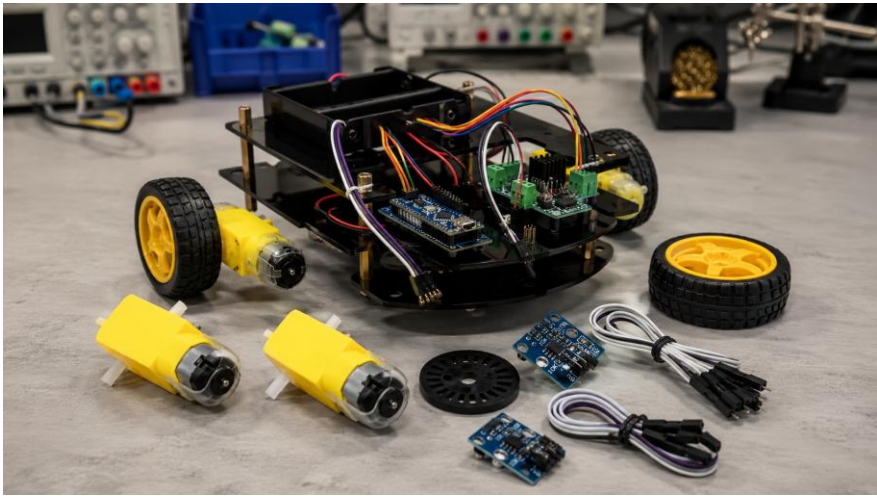


Figura 15.3: Materiales utilizados para implementar odometría con encoders.

15.7. Fundamento teórico

15.7.1. Encoder incremental

Un encoder incremental genera pulsos al girar el eje. Puede ser óptico, magnético o mecánico. En robots móviles se utilizan con frecuencia encoders integrados en motores DC con caja reductora.

Si un encoder entrega N pulsos por revolución de la rueda, cada pulso representa una fracción del giro:

$$\Delta_{rev} = \frac{1}{N}$$

Si se cuenta un total de p pulsos, las revoluciones recorridas son:

$$rev = \frac{p}{N}$$

15.7.2. Distancia recorrida

Si la rueda tiene diámetro D , su perímetro es:

$$C = \pi D$$

La distancia recorrida por la rueda es:

$$d = rev \cdot C$$

Por tanto:

$$d = \frac{p}{N} \cdot \pi D$$

15.7.3. Velocidad y RPM

Si durante un intervalo Δ_t se cuentan Δ_p pulsos, las revoluciones por minuto se calculan como:

$$RPM = \frac{\Delta_p}{N} \cdot \frac{60000}{\Delta_{t_{ms}}}$$

donde $\Delta_{t_{ms}}$ se expresa en milisegundos.

15.7.4. Odometría diferencial

En un robot diferencial, cada rueda puede recorrer una distancia diferente. Si d_L es la distancia recorrida por la rueda izquierda y d_R la distancia recorrida por la derecha, el avance promedio del robot es:

$$d = \frac{d_R + d_L}{2}$$

El cambio aproximado de orientación es:

$$\Delta_\theta = \frac{d_R - d_L}{B}$$

donde B es la distancia entre ruedas.

La posición del robot puede actualizarse como:

$$x_{k+1} = x_k + d \cos\left(\theta_k + \frac{\Delta\theta}{2}\right)$$

$$y_{k+1} = y_k + d \sin\left(\theta_k + \frac{\Delta\theta}{2}\right)$$

$$\theta_{k+1} = \theta_k + \Delta\theta$$

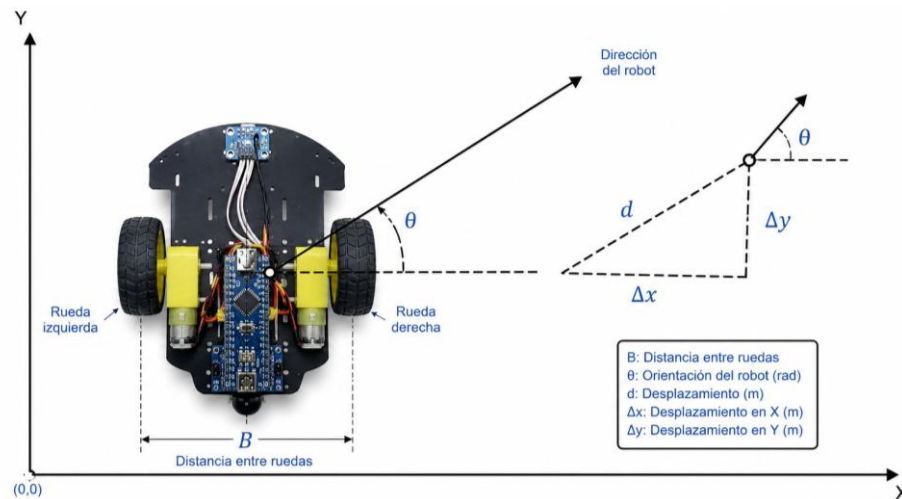


Figura 15.4: Modelo cinemático básico de un robot diferencial.

15.7.5. Error acumulado

La odometría se basa en el movimiento de las ruedas. Si las ruedas patinan, se deforman o atraviesan superficies irregulares, el cálculo puede desviarse de la posición real. Por esta razón, la odometría es útil, pero no perfecta.

Los errores más comunes son:

- Diámetro real de rueda diferente al teórico.
- Pulsos por revolución mal configurados.
- Deslizamiento durante aceleraciones o giros.
- Diferencia mecánica entre motores.
- Superficie irregular.
- Ruedas sucias o con baja adherencia.

15.8. Diseño electrónico

En Arduino Nano existen dos interrupciones externas principales en D2 y D3. Para una versión inicial, se puede usar un canal por encoder y contar pulsos. Si se requiere dirección real con encoder cuadratura, se necesitan más pines y una lógica adicional [1, 2].

Tabla 15.2: Asignación sugerida de pines para odometría y control de motores.

Pin Arduino Nano	Conexión	Función
D2	Encoder izquierdo canal A	Interrupción externa
D3	Encoder derecho canal A	Interrupción externa
D4	AIN1 TB6612FNG	Dirección motor izquierdo
D5	AIN2 TB6612FNG	Dirección motor izquierdo
D6	PWMA TB6612FNG	PWM motor izquierdo
D7	BIN1 TB6612FNG	Dirección motor derecho
D8	BIN2 TB6612FNG	Dirección motor derecho
D9	PWMB TB6612FNG	PWM motor derecho
D10	STBY TB6612FNG	Habilitación del driver
GND	Tierra común	Referencia del sistema

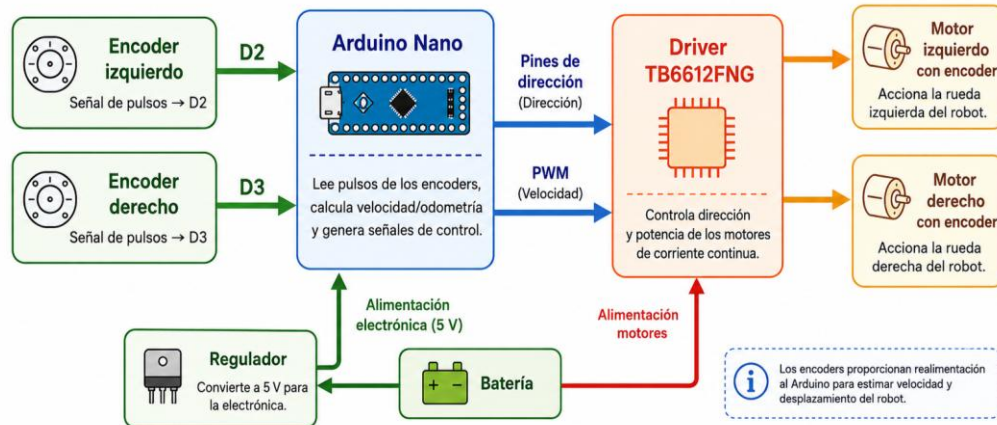


Figura 15.5: Diagrama de conexiones para odometría con encoders.

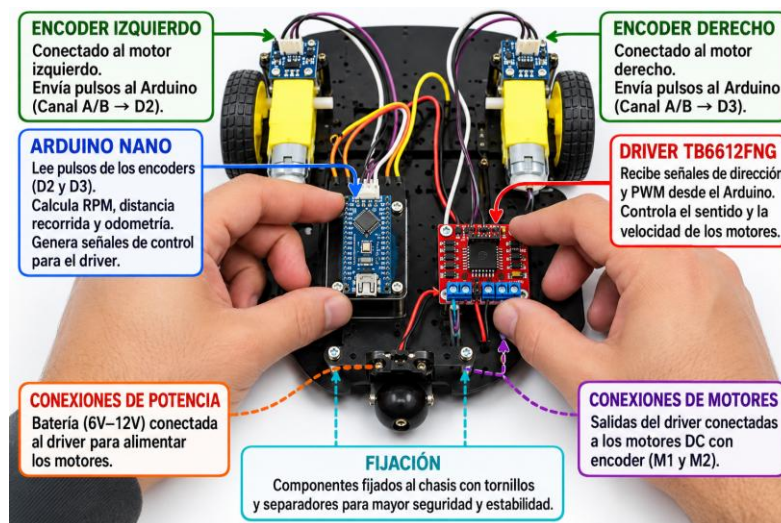


Figura 15.6: Montaje electrónico de encoders y driver de motores.

15.9. Calibración inicial

Antes de usar la odometría, se deben conocer tres parámetros:

- Pulsos por revolución de rueda.
- Diámetro real de rueda.
- Distancia entre ruedas.

Tabla 15.3: Parámetros físicos requeridos para odometría.

Parámetro	Símbolo	Valor medio
Pulsos por revolución	N	
Diámetro de rueda	D	
Distancia entre ruedas	B	
Perímetro de rueda	π	

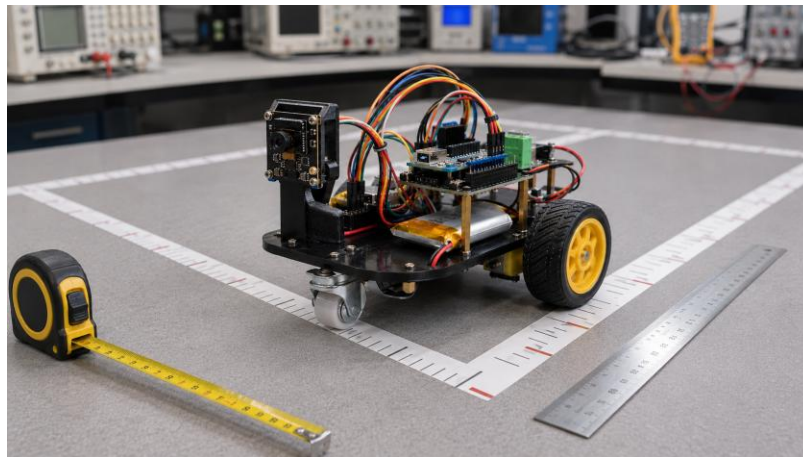


Figura 15.7: Medición de parámetros físicos del robot para odometría.

15.10. Código del proyecto

El siguiente programa lee dos encoders, calcula RPM, distancia recorrida, posición aproximada y aplica un control proporcional de velocidad. Para simplificar, se asume que el sentido de movimiento lo define el comando enviado a los motores [7].

```
// Pegue aquí el código principal del proyecto.
// Mantenga fuente Courier New 9 pt para bloques de código.
/*
Odometria y control de velocidad con encoders
Arduino Nano + TB6612FNG + motores con encoder
*/

// =====
// PINES ENCODERS
```

```

// =====
const byte ENC_IZQ = 2;
const byte ENC_DER = 3;

// =====
// PINES TB6612FNG
// =====
const byte AIN1 = 4;
const byte AIN2 = 5;
const byte PWMA = 6;

const byte BIN1 = 7;
const byte BIN2 = 8;
const byte PWMB = 9;

const byte STBY = 10;

// =====
// PARAMETROS FISICOS
// =====
const float PULSOS_REV = 20.0; // ajustar segun encoder
const float DIAMETRO_RUEDA = 6.5; // cm
const float BASE_RUEDAS = 14.0; // cm

const float PERIMETRO = 3.1416 * DIAMETRO_RUEDA;

// =====
// VARIABLES DE ENCODER
// =====
volatile long pulsosIzq = 0;
volatile long pulsosDer = 0;

long pulsosIzqAnt = 0;
long pulsosDerAnt = 0;

// Variables de velocidad
float rpmIzq = 0;
float rpmDer = 0;

// Odometria
float x = 0;
float y = 0;
float theta = 0;

float distanciaIzqTotal = 0;
float distanciaDerTotal = 0;

```

```

// Control de velocidad
float rpmObjetivoIzq = 60;
float rpmObjetivoDer = 60;

float KpVel = 2.0;

int pwmIzq = 0;
int pwmDer = 0;

int sentidoIzq = 1;
int sentidoDer = 1;

unsigned long tiempoAnterior = 0;
const unsigned long INTERVALO = 100;

// =====
// INTERRUPCIONES
// =====
void contarIzq()
{
    pulsosIzq++;
}

void contarDer()
{
    pulsosDer++;
}

// =====
// MOTORES
// =====
void motorIzquierdo(int pwm)
{
    pwm = constrain(pwm, -255, 255);

    if (pwm > 0)
    {
        sentidoIzq = 1;
        digitalWrite(AIN1, HIGH);
        digitalWrite(AIN2, LOW);
        analogWrite(PWMA, pwm);
    }
    else if (pwm < 0)
    {
        sentidoIzq = -1;
        digitalWrite(AIN1, LOW);
        digitalWrite(AIN2, HIGH);
    }
}

```

```

    analogWrite(PWMA, -pwm);
}
else
{
    digitalWrite(AIN1, LOW);
    digitalWrite(AIN2, LOW);
    analogWrite(PWMA, 0);
}
}

void motorDerecho(int pwm)
{
    pwm = constrain(pwm, -255, 255);

    if (pwm > 0)
    {
        sentidoDer = 1;
        digitalWrite(BIN1, HIGH);
        digitalWrite(BIN2, LOW);
        analogWrite(PWMB, pwm);
    }
    else if (pwm < 0)
    {
        sentidoDer = -1;
        digitalWrite(BIN1, LOW);
        digitalWrite(BIN2, HIGH);
        analogWrite(PWMB, -pwm);
    }
    else
    {
        digitalWrite(BIN1, LOW);
        digitalWrite(BIN2, LOW);
        analogWrite(PWMB, 0);
    }
}

void moverPWM(int izq, int der)
{
    motorIzquierdo(izq);
    motorDerecho(der);
}

// =====
// CALCULO DE ODOMETRIA
// =====
void actualizarOdometria(unsigned long dt)
{

```

```

long pIzq = pulsosIzq;
long pDer = pulsosDer;

long deltaIzq = pIzq - pulsosIzqAnt;
long deltaDer = pDer - pulsosDerAnt;

pulsosIzqAnt = pIzq;
pulsosDerAnt = pDer;

float revIzq = deltaIzq / PULSOS_REV;
float revDer = deltaDer / PULSOS_REV;

rpmIzq = revIzq * (60000.0 / dt);
rpmDer = revDer * (60000.0 / dt);

float dIzq = revIzq * PERIMETRO * sentidoIzq;
float dDer = revDer * PERIMETRO * sentidoDer;

distanciaIzqTotal += dIzq;
distanciaDerTotal += dDer;

float dCentro = (dDer + dIzq) / 2.0;
float dTheta = (dDer - dIzq) / BASE_RUEDAS;

x += dCentro * cos(theta + dTheta / 2.0);
y += dCentro * sin(theta + dTheta / 2.0);
theta += dTheta;
}

// =====
// CONTROL DE VELOCIDAD
// =====
void controlVelocidad()
{
    float errorIzq = rpmObjetivoIzq - rpmIzq;
    float errorDer = rpmObjetivoDer - rpmDer;

    pwmIzq += KpVel * errorIzq;
    pwmDer += KpVel * errorDer;

    pwmIzq = constrain(pwmIzq, -255, 255);
    pwmDer = constrain(pwmDer, -255, 255);

    moverPWM(pwmIzq, pwmDer);
}

// =====

```



```

// COMANDOS POR SERIAL
// =====
void procesarSerial()
{
  if (Serial.available())
  {
    char c = Serial.read();

    if (c == 'F')
    {
      rpmObjetivoIzq = 70;
      rpmObjetivoDer = 70;
    }
    else if (c == 'B')
    {
      rpmObjetivoIzq = -50;
      rpmObjetivoDer = -50;
    }
    else if (c == 'L')
    {
      rpmObjetivoIzq = -45;
      rpmObjetivoDer = 45;
    }
    else if (c == 'R')
    {
      rpmObjetivoIzq = 45;
      rpmObjetivoDer = -45;
    }
    else if (c == 'S')
    {
      rpmObjetivoIzq = 0;
      rpmObjetivoDer = 0;
      pwmIzq = 0;
      pwmDer = 0;
      moverPWM(0, 0);
    }
    else if (c == 'Z')
    {
      x = 0;
      y = 0;
      theta = 0;
      distanciaIzqTotal = 0;
      distanciaDerTotal = 0;
    }
  }
}

```

```

// =====
// TELEMETRIA
// =====
void enviarDatos()
{
    Serial.print("RPM_I:");
    Serial.print(rpmIzq, 1);

    Serial.print(" RPM_D:");
    Serial.print(rpmDer, 1);

    Serial.print(" X:");
    Serial.print(x, 2);

    Serial.print(" Y:");
    Serial.print(y, 2);

    Serial.print(" TH:");
    Serial.print(theta, 3);

    Serial.print(" PWM_I:");
    Serial.print(pwmIzq);

    Serial.print(" PWM_D:");
    Serial.println(pwmDer);
}

// =====
// SETUP Y LOOP
// =====
void setup()
{
    pinMode(ENC_IZQ, INPUT_PULLUP);
    pinMode(ENC_DER, INPUT_PULLUP);

    pinMode(AIN1, OUTPUT);
    pinMode(AIN2, OUTPUT);
    pinMode(PWMA, OUTPUT);

    pinMode(BIN1, OUTPUT);
    pinMode(BIN2, OUTPUT);
    pinMode(PWMB, OUTPUT);

    pinMode(STBY, OUTPUT);
    digitalWrite(STBY, HIGH);

    attachInterrupt(digitalPinToInterrupt(ENC_IZQ), contarIzq, RISING);
}

```

```

attachInterrupt(digitalPinToInterrupt(ENC_DER), contarDer, RISING);

Serial.begin(115200);

tiempoAnterior = millis();

moverPWM(0, 0);

Serial.println("Sistema de odometria iniciado");
Serial.println("Comandos: F avanzar, B retroceder, L izquierda, R derecha, S
detener, Z reset odometria");
}

void loop()
{
  procesarSerial();

  unsigned long ahora = millis();

  if (ahora - tiempoAnterior >= INTERVALO)
  {
    unsigned long dt = ahora - tiempoAnterior;
    tiempoAnterior = ahora;

    actualizarOdometria(dt);
    controlVelocidad();
    enviarDatos();
  }
}

```

El código utiliza un canal por encoder. Para medir dirección real de giro con mayor precisión se recomienda utilizar encoders en cuadratura y leer los canales A y B de cada motor.

15.11. Validación experimental

15.11.1. Prueba 1: conteo de pulsos

Levantar el robot, girar una rueda manualmente y verificar que los pulsos aumenten en el monitor serial.

15.11.2. Prueba 2: una revolución

Marcar una rueda, girarla una vuelta completa y verificar cuántos pulsos cuenta el sistema. Este valor debe coincidir con PULSOS_REV.

15.11.3. Prueba 3: distancia recta

Ordenar al robot avanzar y medir la distancia real con una cinta métrica. Comparar con la distancia calculada por odometría.

15.11.4. Prueba 4: giro sobre su eje

Ordenar un giro izquierda o derecha y verificar si el ángulo estimado cambia de manera coherente.

15.11.5. Prueba 5: control de velocidad

Aplicar diferentes objetivos de RPM y observar si el PWM se ajusta para alcanzar la velocidad deseada.

Tabla 15.4: Registro sugerido de pruebas de odometría.

Prueba	Valor real	Valor estimado	Error
Una revolución rueda izq.			
Una revolución rueda der.			
Avance 50 cm			
Avance 100 cm			
Giro aproximado 90°			

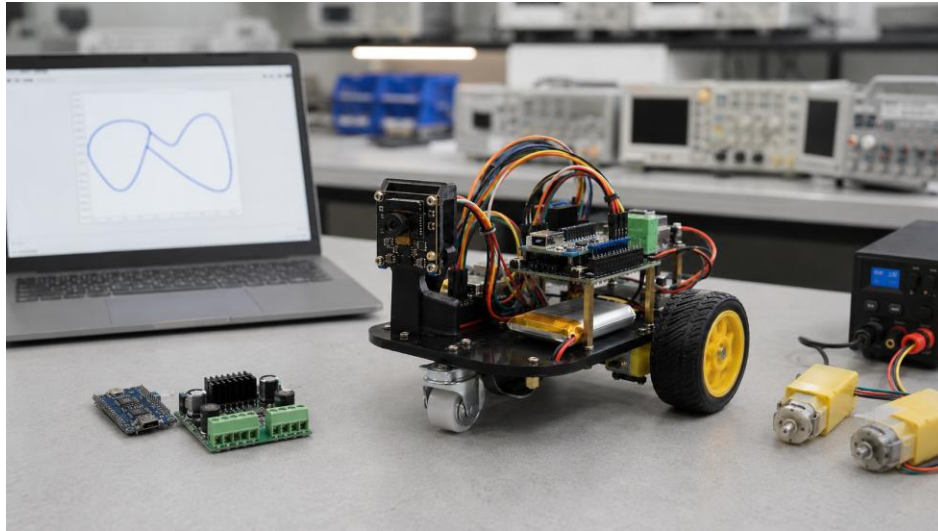


Figura 15.8: Validación experimental de distancia recorrida mediante odometría.

15.12. Ajuste del control de velocidad

El controlador proporcional de velocidad modifica el PWM según el error entre RPM objetivo y RPM medida [6].

Si el robot responde lento, puede aumentarse K_p . Si oscila o cambia bruscamente el PWM, debe reducirse. Para un control más avanzado, puede incorporarse un controlador PI o PID de velocidad.

Tabla 15.5: Guía básica de ajuste del control de velocidad.

Comportamiento	Posible causa	Ajuste sugerido
No alcanza RPM objetivo	K_p bajo o batería débil	Subir K_p o revisar alimentación
PWM cambia bruscamente	K_p alto	Reducir K_p
Un motor va más rápido	Diferencia mecánica	Usar control por rueda
Distancia estimada incorrecta	Diámetro/pulsos mal configurados	Recalibrar parámetros
Giro estimado incorrecto	Base de ruedas incorrecta	Medir B nuevamente

15.13. Integración con robot explorador autónomo

Anteriormente, las maniobras de retroceso y giro se realizaron por tiempo. Con odometría, estas acciones pueden hacerse por distancia o ángulo aproximado.

Por ejemplo:

- Retroceder 10 cm en lugar de retroceder 350 ms.
- Girar hasta que θ cambie 90 grados.
- Avanzar 50 cm y volver a escanear.
- Registrar distancia total recorrida durante una exploración.

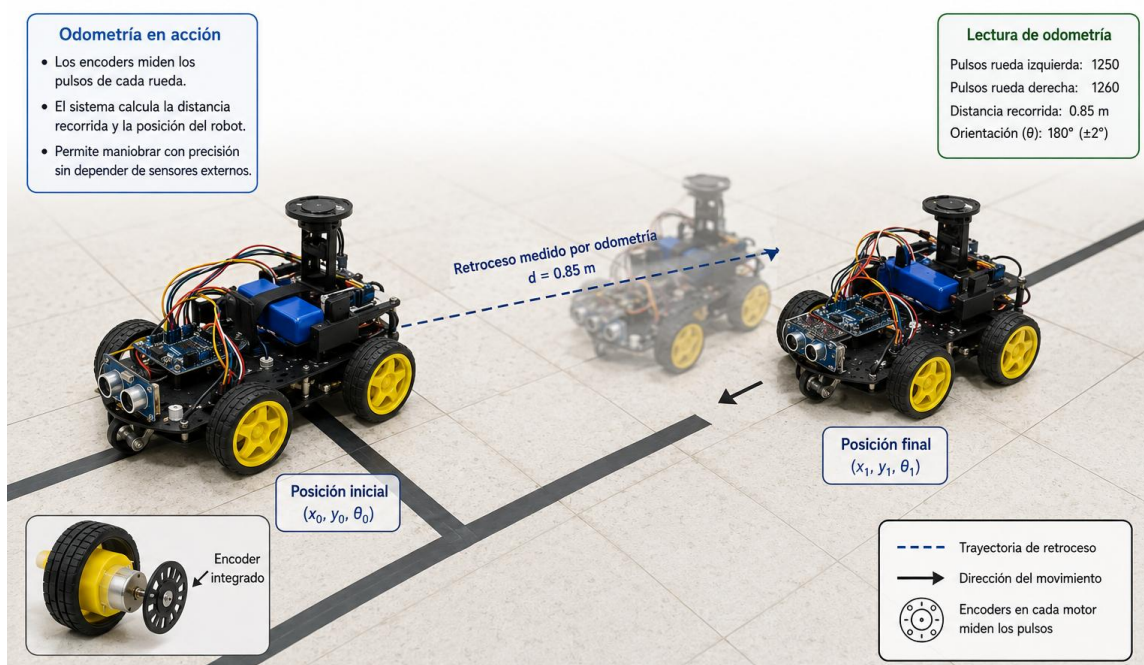


Figura 15.9: Integración de odometría en el robot explorador autónomo.

15.14. Limitaciones de la odometría

Aunque la odometría mejora el control, no debe considerarse una medición absoluta de posición. Los errores se acumulan con el tiempo. En recorridos largos, la posición estimada puede alejarse significativamente de la posición real.

Para mejorar la precisión se pueden combinar encoders con:

- IMU para estimar orientación.
- Sensores de distancia para corregir contra paredes.
- Cámara para reconocer marcas visuales.
- Líneas o referencias en el piso.
- Fusión de sensores mediante filtros.

La odometría es una excelente base para navegación, pero debe complementarse con sensores externos cuando se requiere precisión en recorridos largos.

15.15. Errores frecuentes

Durante el desarrollo pueden presentarse los siguientes errores:

- Configurar incorrectamente pulsos por revolución.
- Medir el diámetro nominal y no el diámetro real de la rueda.
- No usar interrupciones y perder pulsos a alta velocidad.
- Cables de encoder pasando junto a cables de motor sin cuidado.
- No compartir tierra entre encoders, Arduino y driver.
- Usar ruedas con poca adherencia y generar deslizamiento.
- Asumir que odometría y posición real siempre coinciden.

15.16. Proyecto de extensión

Como extensión, se propone implementar movimiento por objetivos:

- Avanzar una distancia específica.
- Girar un ángulo definido.
- Recorrer un cuadrado.
- Regresar aproximadamente al punto de inicio.

También puede integrarse con el dashboard de telemetría para mostrar x , y , θ , RPM y distancia total en una página web.

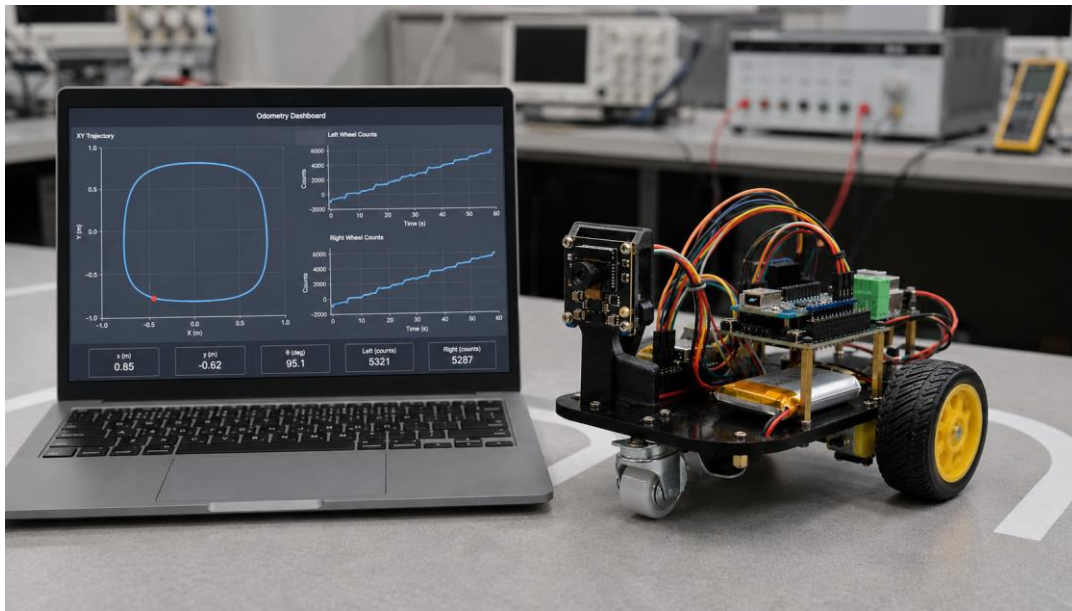


Figura 15.10: Extensión de odometría con dashboard de telemetría.

Actividades de aprendizaje

1. Mida los pulsos por revolución reales de cada encoder.
2. Calcule el perímetro real de la rueda y actualice el código.
3. Compare distancia estimada y distancia real en recorridos de 50 cm y 100 cm.
4. Ajuste K_p del control de velocidad para mejorar la respuesta.
5. Implemente una función para avanzar exactamente 30 cm.
6. Proponga una integración de odometría con el robot explorador.

Preguntas de reflexión

1. ¿Por qué el control por tiempo es menos preciso que el control con encoders?
2. ¿Qué ocurre si el robot patina durante un giro?
3. ¿Por qué es importante medir el diámetro real de las ruedas?
4. ¿Qué ventaja ofrece controlar RPM de cada motor por separado?
5. ¿Por qué la odometría acumula error con el tiempo?
6. ¿Cómo combinaría odometría con ultrasonido o cámara para mejorar navegación?

Referencias

- [1] Arduino, “attachInterrupt()”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/language-reference/en/functions/external-interrupts/attachInterrupt/>. [Accedido: 24-jun-2026].

- [2] Arduino, “digitalPinToInterrupt()”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/language-reference/en/functions/external-interrupts/digitalPinToInterrupt/>. [Accedido: 24-jun-2026].
- [3] Toshiba Electronic Devices & Storage Corporation, “TB6612FNG Brush Motor Driver IC”, Toshiba. [En línea]. Disponible en: <https://toshiba.semicon-storage.com/ap-en/semiconductor/product/motor-driver-ics/brushed-dc-motor-driver-ics/detail.TB6612FNG.html>. [Accedido: 24-jun-2026].
- [4] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.
- [5] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed. Cham, Switzerland: Springer, 2016.
- [6] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ, USA: Princeton University Press, 2008.
- [7] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2017.

Capítulo 16. Navegación por Objetivos con Odometría e IMU

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender la diferencia entre control manual, evasión reactiva y navegación por objetivos.
- Utilizar odometría e IMU para ejecutar movimientos medidos.
- Programar funciones para avanzar una distancia específica.
- Programar funciones para girar hacia un ángulo objetivo.
- Diseñar una máquina de estados para recorrer una secuencia de puntos.
- Implementar navegación básica por coordenadas en un robot diferencial.
- Evaluar errores de trayectoria y proponer mejoras mediante calibración y fusión de sensores.

16.1. Introducción

En los capítulos anteriores se construyeron varias capacidades fundamentales para robots móviles. Primero se implementó control manual por WiFi, luego evasión autónoma de obstáculos, posteriormente odometría con encoders. Cada una de estas capacidades representa un avance en la autonomía del robot.

Sin embargo, para que un robot pueda realizar tareas de navegación más estructuradas, no basta con avanzar o evitar obstáculos. El robot debe ser capaz de ejecutar movimientos medidos, como avanzar 50 cm, girar 90 grados, recorrer un cuadrado o dirigirse hacia un punto aproximado dentro de un plano de trabajo.

Este capítulo desarrolla un sistema de navegación por objetivos utilizando odometría e IMU. El enfoque es educativo y práctico: se parte de un robot diferencial con encoders y orientación fusionada, y se implementan funciones para movimiento por distancia, giro por ángulo y seguimiento de una secuencia de puntos [4].

El proyecto no pretende implementar navegación profesional ni localización absoluta. En lugar de ello, introduce los fundamentos necesarios para que el lector comprenda cómo un robot puede usar mediciones internas para ejecutar trayectorias básicas. Esta base es útil para robots exploradores, plataformas educativas, seguidores avanzados, robots de servicio a escala y demostraciones de robótica inteligente [1, 2].

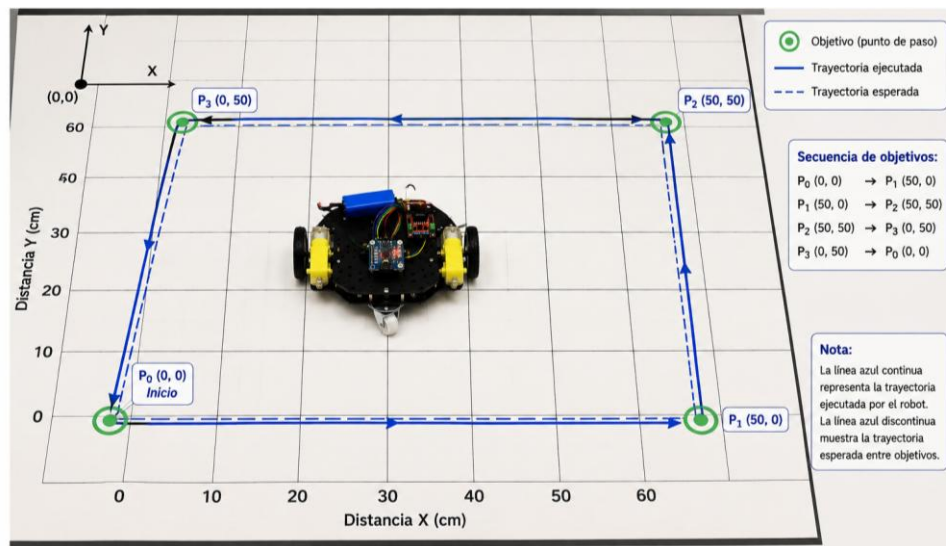


Figura 16.1: Robot móvil ejecutando navegación por objetivos sobre una super cie marcada.

16.2. Planteamiento del problema

Un robot con evasión de obstáculos puede moverse sin chocar, pero no necesariamente llega a un lugar específico. Un robot controlado manualmente puede llegar a un objetivo, pero depende del operador. Un robot con odometría puede medir movimiento, pero necesita una estrategia para convertir esa medición en decisiones de navegación.

El problema central de este capítulo es:

¿Cómo hacer que un robot avance distancias, gire ángulos y recorra objetivos usando sus propias mediciones?

Algunos retos asociados son:

- La odometría acumula error.
- El giroscopio puede presentar deriva.
- Las ruedas pueden patinar.
- El robot puede desviarse durante avance recto.
- Los giros no siempre son exactos.
- La trayectoria real puede diferir de la trayectoria planificada.

Por esta razón, la navegación por objetivos debe diseñarse como una aproximación experimental, con calibración, tolerancias y pruebas sistemáticas.

La navegación por objetivos no significa que el robot conozca perfectamente su posición. Significa que usa sus mediciones para aproximarse a una meta con un error aceptable.

16.3. Descripción general del proyecto

El sistema propuesto utiliza:

- Encoders para medir distancia recorrida.
- MPU6050 para mejorar la estimación de orientación.
- Arduino Nano como controlador.
- Driver TB6612FNG para motores.
- Máquina de estados para ejecutar objetivos.

El robot podrá ejecutar tres tipos de acciones:

1. **Avanzar distancia:** desplazarse una cantidad definida en centímetros.
2. **Girar ángulo:** cambiar orientación hasta alcanzar un ángulo objetivo.
3. **Ir a punto:** calcular dirección y distancia hacia una coordenada objetivo.

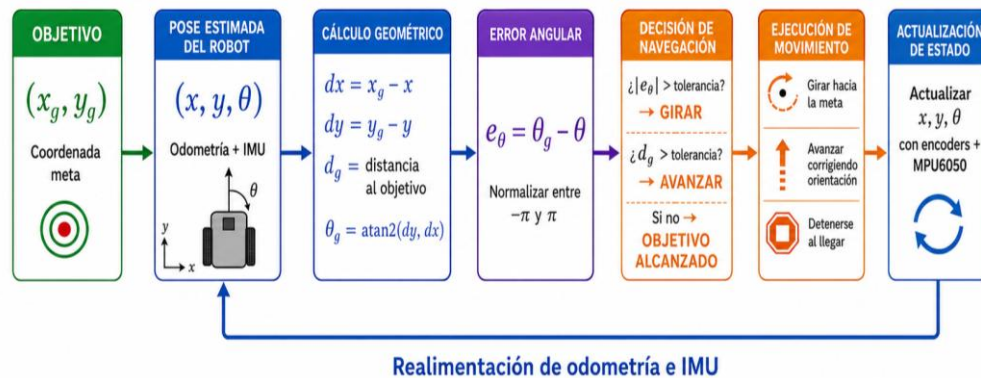


Figura 16.2: Arquitectura de navegación por objetivos.

16.4. Aplicación STEM

Este proyecto integra las áreas STEM de la siguiente manera:

- **Ciencia:** movimiento, orientación, error acumulado, fricción y dinámica del robot.
- **Tecnología:** encoders, IMU, microcontroladores, drivers de motor y sensores inerciales.
- **Ingeniería:** diseño de navegación por objetivos en un robot real.
- **Matemáticas:** coordenadas, trigonometría, distancia euclidiana, error angular y tolerancias.

16.5. Objetivos del proyecto

16.5.1. Objetivo general

Diseñar e implementar un sistema de navegación por objetivos para un robot diferencial, utilizando odometría, orientación fusionada y una máquina de estados para ejecutar movimientos medidos.

16.5.2. *Objetivos específicos*

- 1. Implementar una función para avanzar una distancia definida.
- 2. Implementar una función para girar hacia un ángulo objetivo.
- 3. Calcular distancia y dirección hacia un punto objetivo.
- 4. Diseñar una máquina de estados para recorrer una secuencia de puntos.
- 5. Evaluar el error de posición final.
- 6. Proponer mejoras mediante calibración, reducción de velocidad y corrección de orientación.

16.6. **Materiales y herramientas**

Tabla 16.1: *Materiales principales para navegación por objetivos.*

Elemento	Cantidad	Función
Arduino Nano	1	Control principal
Motores DC con encoder	2	Movimiento y medición
MPU6050	1	Estimación de orientación
Driver TB6612FNG	1	Control de motores
Chasis diferencial	1	Plataforma móvil
Ruedas de buena adherencia	2	Movimiento estable
Batería y regulador	1	Alimentación
Cinta métrica o regla	1	Validación de distancias
Transportador o plantilla angular	1	Validación de giros
Piso o pista cuadriculada	1	Pruebas de navegación

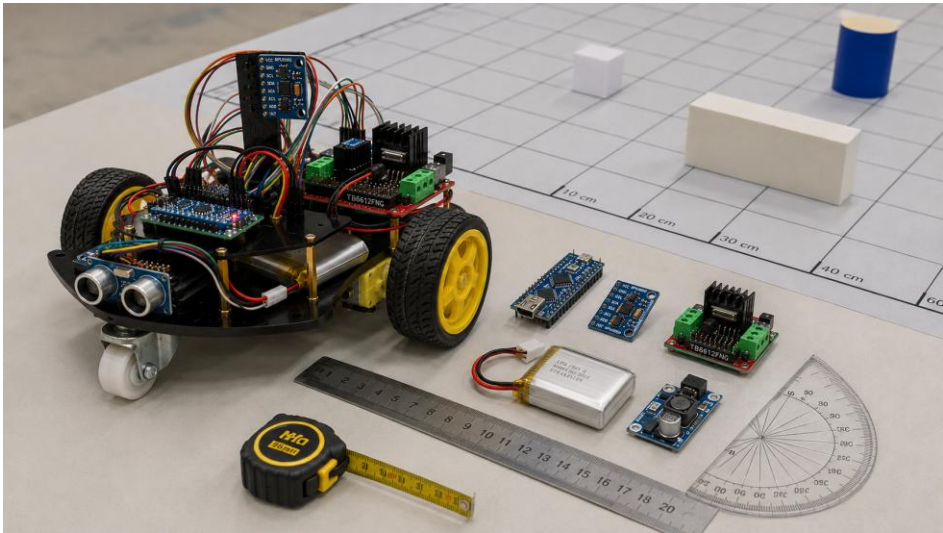


Figura 16.3: *Materiales y entorno de prueba para navegación por objetivos.*

16.7. Fundamento teórico

16.7.1. Distancia hacia un objetivo

Si el robot se encuentra en (x, y) y el objetivo está en (x_g, y_g) , la distancia al objetivo se calcula como:

$$d_g = \sqrt{(x_g - x)^2 + (y_g - y)^2}$$

16.7.2. Ángulo hacia un objetivo

El ángulo deseado se calcula mediante:

$$\theta_g = \tan^{-1} \left(\frac{y_g - y}{x_g - x} \right)$$

En programación se utiliza normalmente $\text{atan2}()$, porque considera correctamente el cuadrante:

$$\theta_g = \text{atan2}(y_g - y, x_g - x)$$

16.7.3. Error angular

El error angular es la diferencia entre el ángulo deseado y la orientación actual [5]:

$$e_\theta = \theta_g - \theta$$

Este error debe normalizarse para quedar entre $-\pi$ y π . De esta forma, el robot gira por el camino más corto.

$$e_\theta \in [-\pi, \pi]$$

16.7.4. Tolerancias

En un robot real, no se debe exigir error cero. Se definen tolerancias:

- Tolerancia de distancia: por ejemplo, 3 cm.
- Tolerancia angular: por ejemplo, 5 grados.

Cuando el robot está dentro de la tolerancia, se considera que alcanzó el objetivo.

Un objetivo no debe entenderse como un punto matemático perfecto. En robótica móvil educativa, conviene definir una zona de llegada.

16.7.1. Estrategia de navegación por punto [3]

La estrategia básica hacia un punto es:

1. Calcular el ángulo hacia el objetivo.
2. Girar hasta quedar orientado hacia ese ángulo.
3. Avanzar mientras se corrige orientación.
4. Detenerse cuando la distancia al objetivo sea menor que la tolerancia.

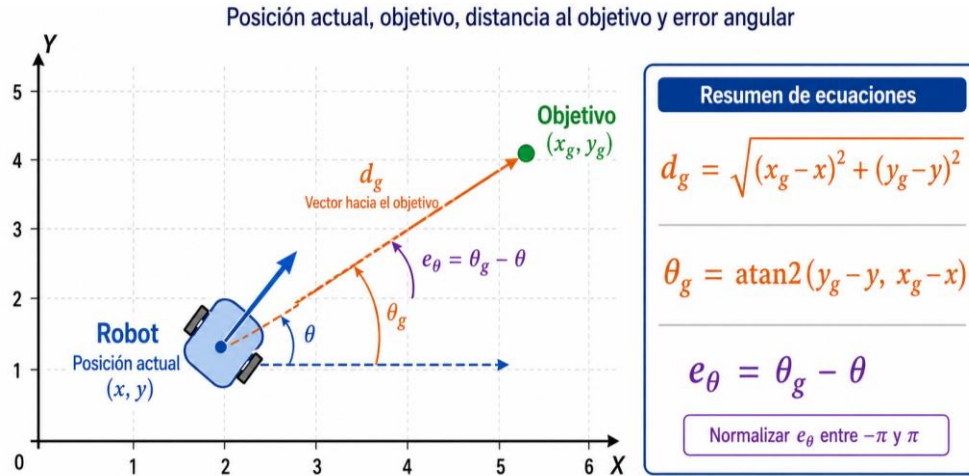


Figura 16.4: Geometría básica para navegación hacia un punto objetivo.

16.8. Diseño electrónico

El diseño electrónico es similar al utilizado en los capítulos anteriores. Se emplean encoders, MPU6050 y driver TB6612FNG [6, 7].

Tabla 16.2: Asignación sugerida de pines para navegación por objetivos.

Pin Arduino Nano	Conexión	Función
D2	Encoder izquierdo	Interrupción externa
D3	Encoder derecho	Interrupción externa
A4	SDA MPU6050	Comunicación I2C
A5	SCL MPU6050	Comunicación I2C
D4	AIN1 TB6612FNG	Dirección motor izquierdo
D5	AIN2 TB6612FNG	Dirección motor izquierdo
D6	PWMA TB6612FNG	PWM motor izquierdo
D7	BIN1 TB6612FNG	Dirección motor derecho
D8	BIN2 TB6612FNG	Dirección motor derecho
D9	PWMB TB6612FNG	PWM motor derecho
D10	STBY TB6612FNG	Habilitación del driver

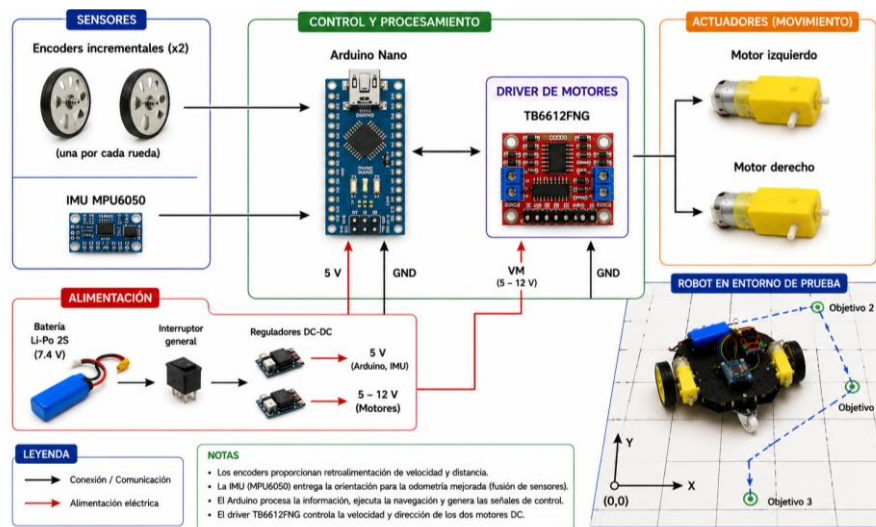


Figura 16.5: Diagrama de conexiones para navegación por objetivos.

16.9. Diseño de la máquina de estados

La navegación se organiza mediante estados [8]:

- **INICIO:** reinicia odometría y carga el primer objetivo.
- **CALCULAR:** calcula distancia y ángulo hacia el objetivo.
- **GIRAR:** orienta el robot hacia el objetivo.
- **AVANZAR:** avanza corrigiendo orientación.
- **SIGUIENTE:** pasa al siguiente objetivo.
- **FIN:** detiene el robot al completar la trayectoria.

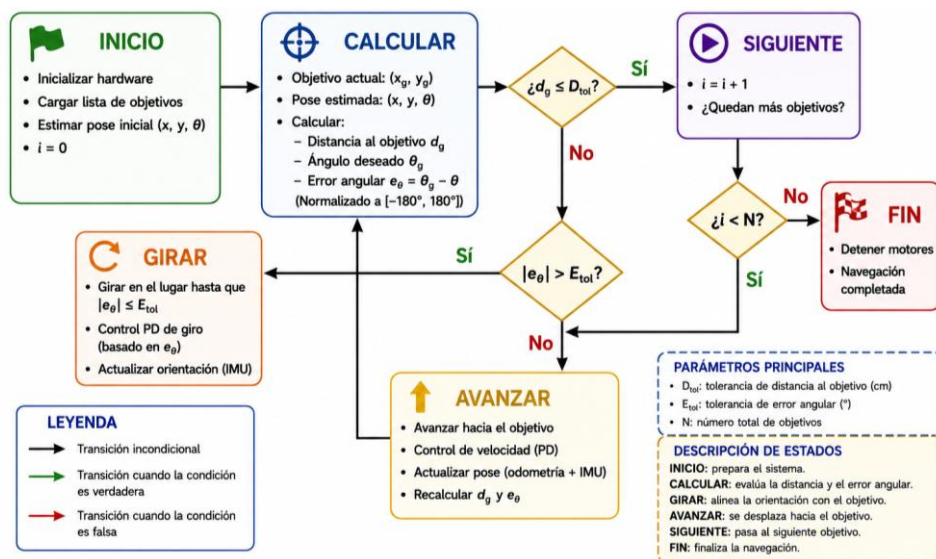


Figura 16.6: Máquina de estados para navegación por objetivos.

16.10. Código del proyecto

El siguiente código presenta una estructura educativa para navegación por objetivos. Integra conceptos de odometría y orientación fusionada, pero se enfoca en la lógica de movimiento hacia puntos. Para mantener claridad, se utiliza una estructura compacta y parámetros ajustables.

```
#include <Wire.h>

// =====
// PINES ENCODERS
// =====
const byte ENC_IZQ = 2;
const byte ENC_DER = 3;

// =====
// PINES TB6612FNG
// =====
const byte AIN1 = 4;
const byte AIN2 = 5;
const byte PWMA = 6;

const byte BIN1 = 7;
const byte BIN2 = 8;
const byte PWMB = 9;

const byte STBY = 10;

// =====
// MPU6050
// =====
const byte MPU_ADDR = 0x68;
const byte REG_PWR_MGMT_1 = 0x6B;
const byte REG_GYRO_XOUT_H = 0x43;
const float SENS_GYRO = 131.0;

// =====
// PARAMETROS FISICOS
// =====
const float PULSOS_REV = 20.0;
const float DIAMETRO_RUEDA = 6.5; // cm
const float BASE_RUEDAS = 14.0; // cm
const float PERIMETRO = 3.1416 * DIAMETRO_RUEDA;

// =====
// NAVEGACION
// =====
const float ALPHA = 0.96;
```



```

const float TOL_DIST = 3.0; // cm
const float TOL_ANG = 5.0 * 3.1416 / 180.0; // rad

const int PWM_AVANCE = 120; 46
const int PWM_GIRO = 100;

float KpAng = 80.0;

// =====
// OBJETIVOS
// Coordenadas en centimetros
// =====
struct Punto
{
    float x;
    float y;
};

Punto objetivos [] = {
    {50, 0},
    {50, 50},
    {0, 50},
    {0, 0}
};

const byte NUM_OBJETIVOS = sizeof(objetivos) / sizeof(objetivos[0]);
byte objetivoActual = 0;

// =====
// VARIABLES DE ODOMETRIA
// =====
volatile long pulsosIzq = 0;
volatile long pulsosDer = 0;

long pulsosIzqAnt = 0;
long pulsosDerAnt = 0;

float x = 0;
float y = 0;
float thetaEnc = 0;
float thetaFusion = 0;

float biasGyroZ = 0;

unsigned long tiempoAnterior = 0;

// =====

```

```

// MAQUINA DE ESTADOS
// =====
enum EstadoNav
{
    INICIO,
    CALCULAR,
    GIRAR,
    AVANZAR,
    SIGUIENTE,
    FIN
};

EstadoNav estado = INICIO;

float distanciaObjetivo = 0;
float anguloObjetivo = 0;
float errorAngular = 0;

// =====
// INTERRUPCIONES
// =====
void contarIzq()
{
    pulsosIzq++;
}

void contarDer()
{
    pulsosDer++;
}

// =====
// UTILIDADES MATEMATICAS
// =====
float normalizarAngulo(float a)
{
    while (a > 3.1416) a -= 2.0 * 3.1416;
    while (a < -3.1416) a += 2.0 * 3.1416;
    return a;
}

// =====
// MPU6050 132
// =====
void iniciarMPU6050()
{
    Wire.begin();

```

```

Wire.beginTransmission(MPU_ADDR);
Wire.write(REG_PWR_MGMT_1);
Wire.write(0);
Wire.endTransmission(true);
}

int16_t leerWordMPU(byte registro)
{
    Wire.beginTransmission(MPU_ADDR);
    Wire.write(registro);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU_ADDR, (byte)2, true);

    int16_t valor = Wire.read() << 8 | Wire.read();
    return valor;
}

int16_t leerGyroZRaw()
{
    return leerWordMPU(REG_GYRO_XOUT_H + 4);
}

float leerGyroZ_rad_s()
{
    int16_t raw = leerGyroZRaw();
    float gzDeg = (raw - biasGyroZ) / SENS_GYRO;
    return gzDeg * 3.1416 / 180.0;
}

void calibrarGyro()
{
    long suma = 0;
    const int muestras = 1000;

    Serial.println("Calibrando giroscopio. No mover robot.");

    for (int i = 0; i < muestras; i++)
    {
        suma += leerGyroZRaw();
        delay(2);
    }

    biasGyroZ = suma / (float)muestras;

    Serial.print("Bias Gyro Z: ");
    Serial.println(biasGyroZ);
}

```

```

}

// =====
// MOTORES
// =====

void motorIzquierdo(int pwm)
{
    pwm = constrain(pwm, -255, 255);

    if (pwm > 0)
    {
        digitalWrite(AIN1, HIGH);
        digitalWrite(AIN2, LOW);
        analogWrite(PWMA, pwm);
    }
    else if (pwm < 0)
    {
        digitalWrite(AIN1, LOW);
        digitalWrite(AIN2, HIGH);
        analogWrite(PWMA, -pwm);
    }
    else
    {
        digitalWrite(AIN1, LOW);
        digitalWrite(AIN2, LOW);
        analogWrite(PWMA, 0);
    }
}

void motorDerecho(int pwm)
{
    pwm = constrain(pwm, -255, 255);
    if (pwm > 0)
    {
        digitalWrite(BIN1, HIGH);
        digitalWrite(BIN2, LOW);
        analogWrite(PWMB, pwm);
    }
    else if (pwm < 0)
    {
        digitalWrite(BIN1, LOW);
        digitalWrite(BIN2, HIGH);
        analogWrite(PWMB, -pwm);
    }
    else
    {
        digitalWrite(BIN1, LOW);

```

```

    digitalWrite(BIN2, LOW);
    analogWrite(PWMB, 0);
}
}

void mover(int izq, int der)
{
    motorIzquierdo(izq);
    motorDerecho(der);
}

void detener()
{
    mover(0, 0);
}

// =====
// ACTUALIZACION DE ODOMETRIA Y FUSION
// =====
void actualizarPose(float dt)
{
    long pIzq = pulsosIzq;
    long pDer = pulsosDer;

    long deltaIzq = pIzq - pulsosIzqAnt;
    long deltaDer = pDer - pulsosDerAnt;

    pulsosIzqAnt = pIzq;
    pulsosDerAnt = pDer;

    float dIzq = (deltaIzq / PULSOS_REV) * PERIMETRO;
    float dDer = (deltaDer / PULSOS_REV) * PERIMETRO;

    float dCentro = (dDer + dIzq) / 2.0;

    float dThetaEnc = (dDer - dIzq) / BASE_RUEDAS;
    thetaEnc += dThetaEnc;

    float gyroZ = leerGyroZ_rad_s();

    thetaFusion = ALPHA * (thetaFusion + gyroZ * dt) +
        (1.0 - ALPHA) * thetaEnc;
    thetaFusion = normalizarAngulo(thetaFusion);
    x += dCentro * cos(thetaFusion);
    y += dCentro * sin(thetaFusion);
}

```

```

// =====
// CALCULO DE OBJETIVO
// =====
void calcularObjetivo()
{
    Punto meta = objetivos[objetivoActual];

    float dx = meta.x - x;
    float dy = meta.y - y;

    distanciaObjetivo = sqrt(dx * dx + dy * dy);
    anguloObjetivo = atan2(dy, dx);

    errorAngular = normalizarAngulo(anguloObjetivo - thetaFusion);
}

// =====
// ESTADOS
// =====
void ejecutarInicio()
{
    x = 0;
    y = 0;
    thetaEnc = 0;
    thetaFusion = 0;
    pulsosIzq = 0;
    pulsosDer = 0;
    pulsosIzqAnt = 0;
    pulsosDerAnt = 0;

    objetivoActual = 0;
    estado = CALCULAR;
    Serial.println("Navegacion iniciada");
}

void ejecutarCalcular()
{
    calcularObjetivo();

    if (distanciaObjetivo <= TOL_DIST)
    {
        estado = SIGUIENTE;
    }
    else if (abs(errorAngular) > TOL_ANG)
    {
        estado = GIRAR;
    }
}

```

```

else
{
    estado = AVANZAR;
}
}

void ejecutarGirar()
{
    calcularObjetivo();

    if (abs(errorAngular) <= TOL_ANG)
    {
        detener();
        estado = AVANZAR;
        return;
    }

    int pwm = PWM_GIRO;
    if (errorAngular > 0)
    {
        mover(-pwm, pwm);
    }
    else
    {
        mover(pwm, -pwm);
    }
}

void ejecutarAvanzar()
{
    calcularObjetivo();
    if (distanciaObjetivo <= TOL_DIST)
    {
        detener();
        estado = SIGUIENTE;
        return;
    }

    if (abs(errorAngular) > 0.35)
    {
        detener();
        estado = GIRAR;
        return;
    }

    int correccion = KpAng * errorAngular;

```

```

int pwmIzq = PWM_AVANCE - correccion;
int pwmDer = PWM_AVANCE + correccion;

pwmIzq = constrain(pwmIzq, 70, 180);
pwmDer = constrain(pwmDer, 70, 180);

mover(pwmIzq, pwmDer);
}

void ejecutarSiguiente()
{
    objetivoActual++;

    if (objetivoActual >= NUM_OBJETIVOS)
    {
        estado = FIN;
    }
    else
    {
        estado = CALCULAR;
    }
}

void ejecutarFin()
{
    detener();
}

// =====
// TELEMETRIA
// =====
void enviarTelemetria()
{
    Serial.print("Obj:");
    Serial.print(objetivoActual);

    Serial.print(" X:");
    Serial.print(x, 1);

    Serial.print(" Y:");
    Serial.print(y, 1);

    Serial.print(" TH:");
    Serial.print(thetaFusion, 2);

    Serial.print(" Dist:");
    Serial.print(distanciaObjetivo, 1);
}

```



```

Serial.print(" Eang:");
Serial.print(errorAngular, 2);

Serial.print(" Estado:");
Serial.println(estado);
}

// =====
// SETUP Y LOOP
// =====
void setup()
{
  pinMode(ENC_IQZ, INPUT_PULLUP);
  pinMode(ENC_DER, INPUT_PULLUP);

  pinMode(AIN1, OUTPUT);
  pinMode(AIN2, OUTPUT);
  pinMode(PWMA, OUTPUT);

  pinMode(BIN1, OUTPUT);
  pinMode(BIN2, OUTPUT);
  pinMode(PWMB, OUTPUT);

  pinMode(STBY, OUTPUT);
  digitalWrite(STBY, HIGH);

  Serial.begin(115200);

  iniciarMPU6050();
  delay(500);
  calibrarGyro();

  attachInterrupt(digitalPinToInterrupt(ENC_IQZ), contarIzq, RISING);
  attachInterrupt(digitalPinToInterrupt(ENC_DER), contarDer, RISING);

  tiempoAnterior = millis();

  detener();

  estado = INICIO;
}

void loop()
{
  unsigned long ahora = millis();

```

```

if (ahora - tiempoAnterior >= 50)
{
    float dt = (ahora - tiempoAnterior) / 1000.0;
    tiempoAnterior = ahora;

    actualizarPose(dt);

    switch (estado)
    {
        case INICIO:
            ejecutarInicio();
            break;

        case CALCULAR:
            ejecutarCalcular();
            break;

        case GIRAR:
            ejecutarGirar();
            break;

        case AVANZAR:
            ejecutarAvanzar();
            break;

        case SIGUIENTE:
            ejecutarSiguiente();
            break;

        case FIN:
            ejecutarFin();
            break;
    }

    enviarTelemetria();
}
}

```

Este código es una base educativa. Para robots rápidos o trayectorias exigentes se recomienda añadir control de velocidad por encoders, rampas de aceleración, sensores externos y verificación de obstáculos.

16.11. Trayectorias básicas

16.11.1. Trayectoria en línea recta

El primer experimento consiste en un solo objetivo:

$$(50, 0)$$

El robot debe avanzar aproximadamente 50 cm desde el origen. Esta prueba valida distancia, diámetro de rueda, pulsos por revolución y avance recto.

16.11.2. Trayectoria cuadrada

Una prueba más completa consiste en cuatro puntos:

$$(50, 0), (50, 50), (0, 50), (0, 0)$$

El robot debería recorrer un cuadrado y volver cerca del punto inicial. El error final permite evaluar la calidad de la odometría.

16.11.3. Trayectoria triangular

Otra opción es definir:

$$(40, 0), (20, 35), (0, 0)$$

Esta prueba obliga al robot a realizar giros diferentes a 90 grados.

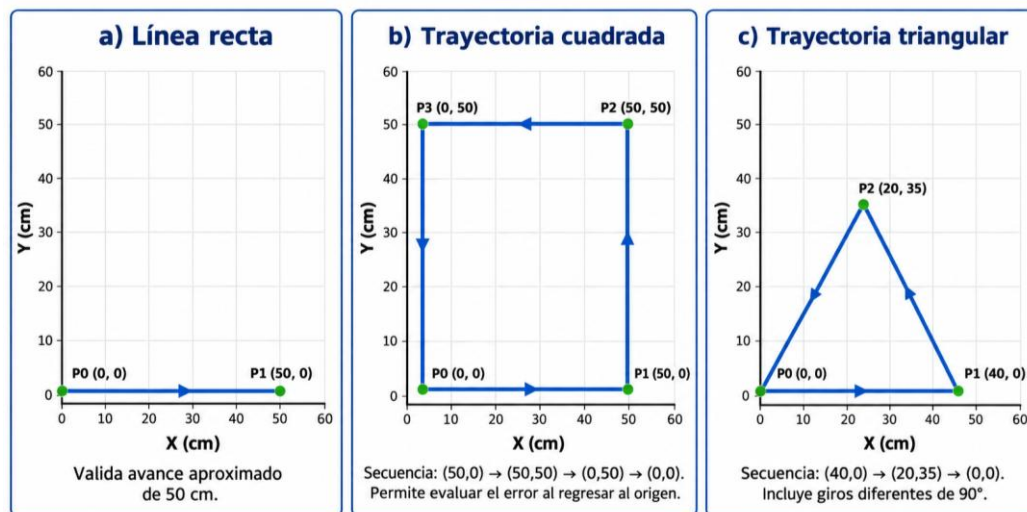


Figura 16.7: Trayectorias básicas para validar navegación por objetivos.

16.12. Validación experimental

16.12.1. Prueba 1: avance a un punto

Definir un objetivo a 50 cm al frente. Medir la distancia real alcanzada.

16.12.2. Prueba 2: giro hacia objetivo lateral

Definir un objetivo a la izquierda o derecha. Verificar que el robot gire primero y avance después.

16.12.3. Prueba 3: trayectoria cuadrada

Ejecutar la trayectoria cuadrada y medir el error final respecto al origen.

16.12.4. Prueba 4: repetibilidad

Ejecutar la misma trayectoria tres veces. Comparar si el error final es similar.

16.12.5. Prueba 5: superficie diferente

Repetir la trayectoria sobre otra superficie y observar el cambio de error.

Tabla 16.3: Registro sugerido de pruebas de navegación por objetivos.

Prueba	Objetivo	Posición real	Error	Observación
Línea recta	(50, 0)			
Cuadrado 1	Retorno a origen			
Cuadrado 2	Retorno a origen			
Triángulo	Retorno a origen			
Otra superficie	Retorno a origen			

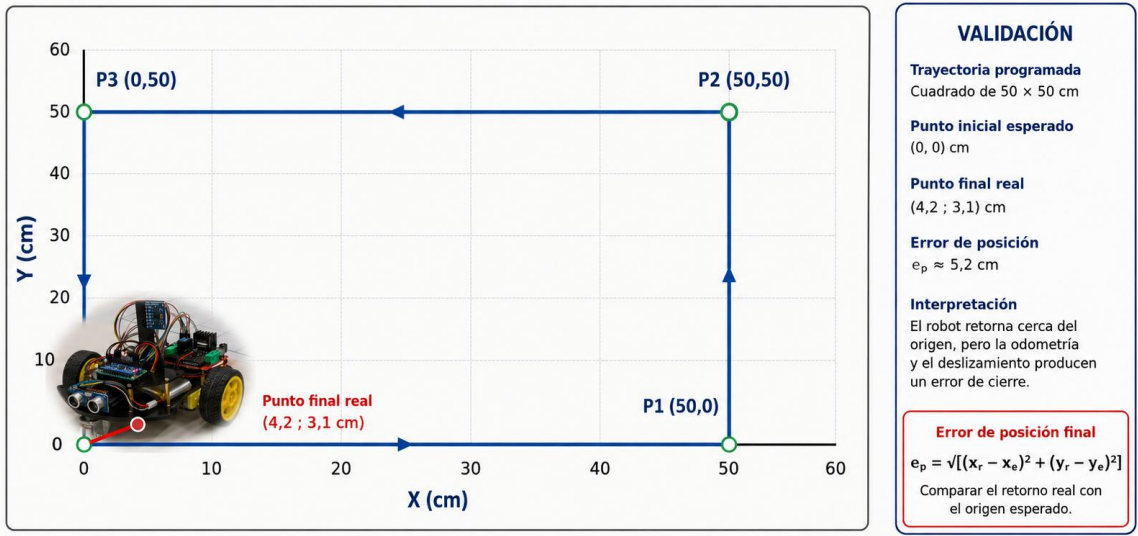


Figura 16.8: Validación experimental de una trayectoria por objetivos.

16.13. Análisis del error

El error puede medirse como distancia entre el punto final real y el punto esperado:

$$e_p = \sqrt{(x_{real} - x_{esp})^2 + (y_{real} - y_{esp})^2}$$

También se puede registrar error angular final:

$$e_\theta = \theta_{real} - \theta_{esp}$$

Al analizar el error, se debe considerar:

- Si el error crece siempre hacia el mismo lado, puede existir desbalance mecánico.
- Si el error cambia mucho entre pruebas, puede haber patinaje o baja repetibilidad.
- Si el robot no cierra el cuadrado, puede estar mal calibrada la distancia entre ruedas.
- Si avanza de más o de menos, puede estar mal calibrado el diámetro de rueda.

16.14. Mejoras recomendadas

Para mejorar la navegación se recomienda:

- Reducir velocidad durante giros.
- Usar rampas de aceleración y frenado.
- Calibrar diámetro real de ruedas.
- Calibrar distancia efectiva entre ruedas.
- Usar control de velocidad por encoders.
- Evitar superficies resbalosas.
- Combinar con sensores de obstáculos.
- Usar referencias externas, como marcas visuales o líneas.

16.15. Integración con el dashboard de telemetría

La navegación por objetivos puede integrarse con un dashboard. El panel podría mostrar:

- Posición estimada x, y .
- Orientación θ .
- Objetivo actual.
- Distancia al objetivo.
- Error angular. Estado de navegación.



Figura 16.9: Extensión con dashboard para navegación por objetivos.

16.16. Limitaciones del sistema

La navegación por objetivos basada solo en odometría e IMU tiene limitaciones:

- No detecta obstáculos inesperados si no se agregan sensores externos.
- La posición estimada acumula error.
- El giroscopio deriva.
- La odometría falla con patinaje.
- No existe corrección absoluta de posición.
- Las trayectorias largas presentan mayor error final.

La navegación por objetivos de este capítulo debe verse como una base. Para navegación robusta se requiere combinar odometría, IMU, sensores de entorno y referencias externas.

16.17. Errores frecuentes

Durante el desarrollo pueden presentarse los siguientes errores:

- No normalizar el error angular entre $-\pi$ y π .
- Confundir grados con radianes.
- Usar tolerancias demasiado pequeñas.
- Intentar navegar a alta velocidad sin control de velocidad.
- No calibrar pulsos por revolución.
- No calibrar distancia entre ruedas.
- No reiniciar la odometría antes de una prueba.
- Esperar precisión absoluta en una plataforma con ruedas y deslizamiento.

16.18. Proyecto de extensión

Como extensión, se propone implementar una lista de trayectorias seleccionables por Bluetooth o WiFi. El usuario podría elegir desde una interfaz:

- Línea recta.
- Cuadrado.
- Triángulo.
- Recorrido personalizado.
- Retorno al origen.

Otra mejora consiste en combinar navegación por objetivos con evasión de obstáculos. Si aparece un obstáculo, el robot podría detener la navegación, ejecutar evasión y luego intentar regresar al objetivo.

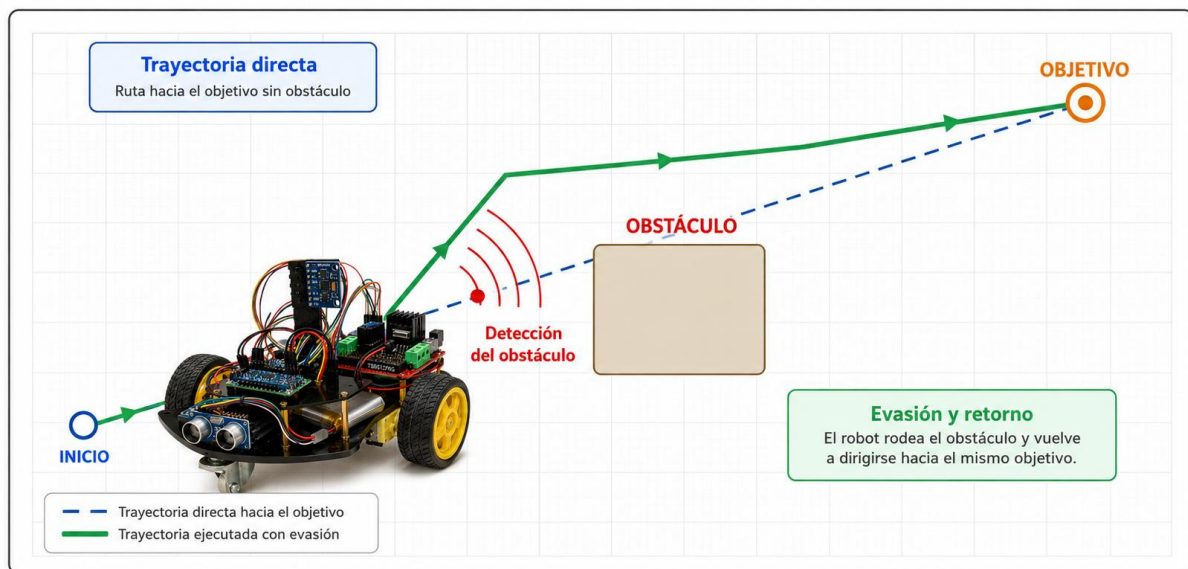


Figura 16.10: Extensión de navegación por objetivos con evasión de obstáculos.

Actividades de aprendizaje

1. Modifique la lista de objetivos para que el robot avance 30 cm y regrese al origen.
2. Diseñe una trayectoria cuadrada de 40 cm por lado.
3. Mida el error final después de tres repeticiones.
4. Ajuste la tolerancia angular y observe el efecto en la trayectoria.
5. Agregue una rampa de aceleración al avance.
6. Proponga una versión con dashboard para visualizar el objetivo actual.

Preguntas de reflexión

1. ¿Por qué es necesario normalizar el error angular?
2. ¿Qué ocurre si la tolerancia de distancia es demasiado pequeña?
3. ¿Por qué una trayectoria cuadrada es útil para validar odometría?
4. ¿Qué error indica una mala calibración de la distancia entre ruedas?
5. ¿Cómo se podría combinar esta navegación con ultrasonido?
6. ¿Qué sensores agregaría para reducir el error acumulado?

Referencias

- [1] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.
- [2] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed. Cham, Switzerland: Springer, 2016.
- [3] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006.
- [4] P. D. Groves, *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems*, 2nd ed. Boston, MA, USA: Artech House, 2013.
- [5] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ, USA: Princeton University Press, 2008.
- [6] Arduino, “attachInterrupt()”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/language-reference/en/functions/external-interrupts/attachInterrupt/>. [Accedido: 24-jun-2026].
- [7] InvenSense, “MPU-6000 and MPU-6050 Product Specification”, Rev. 3.4. [En línea]. Disponible en: <https://cdn.sparkfun.com/datasheets/Components/General%20IC/PS-MPU-6000A.pdf>. [Accedido: 24-jun-2026].
- [8] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2017.

Capítulo 17. Control Difuso para Robots Competitivos

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender los fundamentos del control difuso aplicado a robots competitivos.
- Diferenciar una estrategia basada en condiciones if de una estrategia basada en reglas difusas.
- Diseñar funciones de pertenencia para variables como distancia, error lateral y velocidad.
- Crear reglas lingüísticas para controlar ataque, búsqueda, giro y evasión.
- Implementar un controlador difuso básico en Arduino sin utilizar librerías externas.
- Aplicar control difuso a un robot Minisumo o robot móvil diferencial.
- Evaluar ventajas, limitaciones y criterios de ajuste del control difuso en competencias.

17.1. Introducción

En capítulos anteriores se utilizaron máquinas de estados, PID, odometría, sensores de distancia y fusión sensorial. Estas técnicas permiten que un robot lea su entorno, tome decisiones y controle sus motores. Sin embargo, en competencias reales el comportamiento del robot no siempre puede describirse con condiciones rígidas.

Por ejemplo, en un Minisumo, el oponente puede estar ligeramente a la izquierda, muy cerca, parcialmente detectado o moverse con rapidez. En un seguidor de línea, el error no siempre es simplemente pequeño o grande, sino una transición continua entre varias condiciones. El control difuso permite trabajar con esas zonas intermedias.

En lugar de afirmar únicamente que un objeto está cerca o lejos, se puede expresar que está poco cerca, medianamente cerca o muy cerca. En lugar de decidir que el robot gira o no gira, se puede calcular cuánto debe girar de acuerdo con reglas lingüísticas.

Este capítulo introduce el control difuso desde una perspectiva práctica. El proyecto se orienta a un robot competitivo diferencial, especialmente Minisumo, donde se combinan dos entradas: posición lateral del oponente y proximidad. La salida del sistema será una acción de movimiento expresada como velocidad y giro.

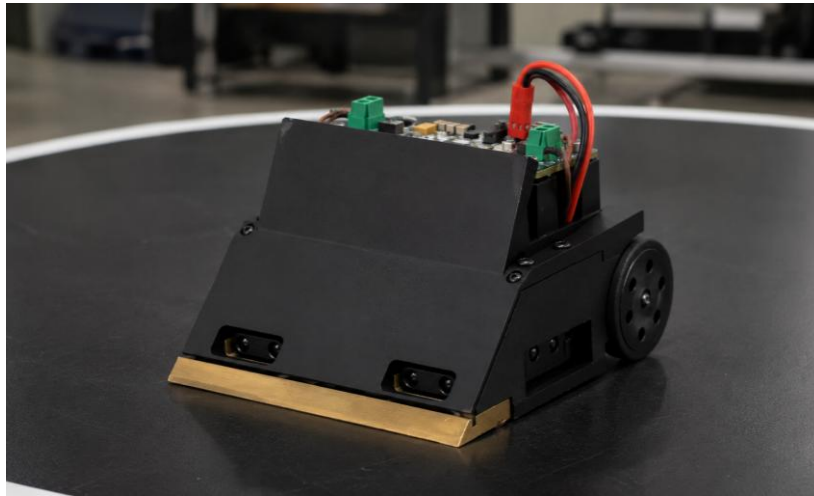


Figura 17.1: Robot competitivo como plataforma de aplicación para control difuso.

17.2. Planteamiento del problema

Las estrategias tradicionales suelen escribirse con estructuras if:

```
if (oponenteCentro) {  
  avanzarRapido();  
} else if (oponenteIzquierda) {  
  girarIzquierda();  
} else if (oponenteDerecha) {  
  girarDerecha();  
} else {  
  buscar();  
}
```

Esta estrategia es sencilla, pero presenta limitaciones: cambia bruscamente de una acción a otra, no representa condiciones intermedias, puede reaccionar de forma agresiva ante ruido y requiere muchos if cuando aumentan los sensores.

El control difuso permite transformar condiciones rígidas en reglas más flexibles, por ejemplo:

Si el oponente está algo a la izquierda y está muy cerca, girar fuerte hacia la izquierda y atacar con velocidad alta.

El control difuso no busca que el robot piense como una persona. Busca convertir decisiones humanas aproximadas en reglas matemáticas implementables.

17.3. Descripción general del proyecto

El proyecto consiste en implementar un controlador difuso para un robot móvil diferencial. El robot utiliza tres sensores frontales de oponente: izquierdo, central y derecho. También puede

incluir sensores de borde, como en un Minisumo. Por seguridad, la evasión de borde tiene prioridad absoluta [5, 6].

Cuando no hay borde, el controlador difuso calcula:

- Error lateral del oponente.
- Proximidad del oponente.
- Intensidad de giro.
- Velocidad de ataque.

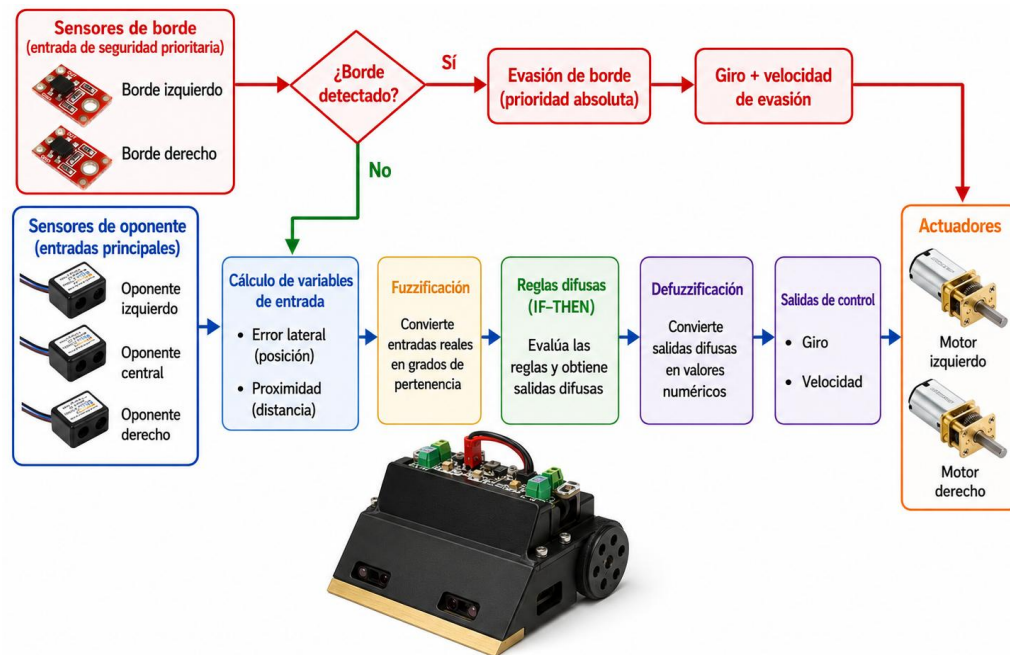


Figura 17.2: Arquitectura general del controlador difuso para robot competitivo.

17.4. Aplicación STEM

Este proyecto integra las áreas STEM de la siguiente manera:

- **Ciencia:** percepción, incertidumbre, ruido y comportamiento dinámico.
- **Tecnología:** sensores, microcontroladores, drivers de motor y programación embebida.
- **Ingeniería:** diseño de reglas de decisión para robots competitivos.
- **Matemáticas:** funciones de pertenencia, inferencia, ponderación y promedios.

17.5. Objetivos del proyecto

17.5.1. Objetivo general

Diseñar e implementar un controlador difuso básico para un robot competitivo diferencial, capaz de ajustar giro y velocidad de ataque en función de la posición y proximidad del oponente.

17.5.2. Objetivos específicos

- 1. Definir variables lingüísticas de entrada y salida.
- 2. Diseñar funciones de pertenencia triangulares y trapezoidales.
- 3. Crear una base de reglas difusas para ataque y búsqueda.
- 4. Implementar inferencia difusa simplificada en Arduino.
- 5. Defuzzificar salidas mediante promedio ponderado.
- 6. Integrar prioridad de seguridad mediante sensores de borde.
- 7. Evaluar el comportamiento frente a diferentes posiciones del oponente.

17.6. Materiales y herramientas

Tabla 17.1: Materiales principales para el proyecto de control difuso.

Elemento	Cantidad	Función
Arduino Nano o compatible	1	Control principal
Sensores de distancia IR o ToF	3	Detección de oponente
Sensores reflectivos de borde	2	Seguridad en dohyo
Driver TB6612FNG o similar	1	Control de motores DC
Motores DC con caja reductora	2	Tracción diferencial
Batería LiPo	1	Alimentación del robot
Chasis Minisumo o robot móvil	1	Plataforma de prueba
Dohyo o pista de prueba	1	Validación experimental

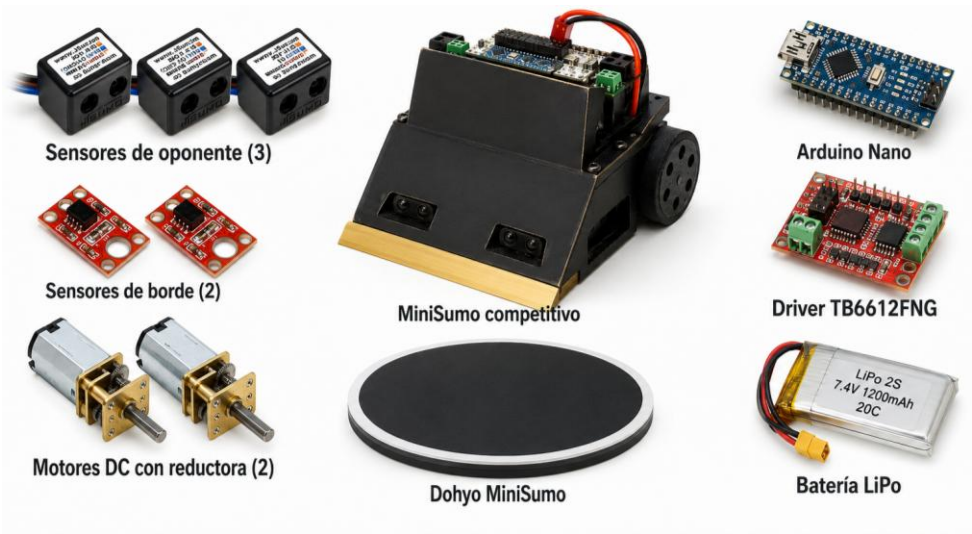


Figura 17.3: Materiales utilizados para implementar control difuso.

17.7. Fundamento teórico

17.7.1. Lógica clásica y lógica difusa

La lógica clásica trabaja con valores verdaderos o falsos. Por ejemplo [1]:

$$distancia < 20\text{ cm} \Rightarrow cerca$$

Si la distancia es 19 cm, el objeto está cerca. Si es 21 cm, ya no lo está. La lógica difusa permite grados de pertenencia:

$$\mu_{cerca}(x) \in [0, 1]$$

Un objeto puede pertenecer al conjunto *cerca* con grado 0.7 y al conjunto *medio* con grado 0.3.

17.7.2. Variables lingüísticas

Una variable lingüística representa una magnitud mediante palabras. Para este proyecto se proponen [2]:

- **Error lateral:** izquierda, centro, derecha.
- **Proximidad:** lejos, medio, cerca.
- **Giro:** izquierda fuerte, recto, derecha fuerte.
- **Velocidad:** baja, media, alta.

17.7.3. Función de pertenencia triangular

Una función triangular se define por tres puntos a, b y c:

$$\begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & a < x \leq b \\ \frac{c-x}{c-b}, & b < x < c \\ 0, & x \geq c \end{cases}$$

17.7.4. Reglas difusas

Una regla tiene la forma:

Si entrada es A y entrada es B, entonces salida es C [3]

Para combinar dos condiciones se puede usar el mínimo:

$$\alpha = \min(\mu_A, \mu_B)$$

Luego, para obtener una salida numérica se usa promedio ponderado:

$$salida = \frac{\sum \alpha_i z_i}{\sum \alpha_i}$$

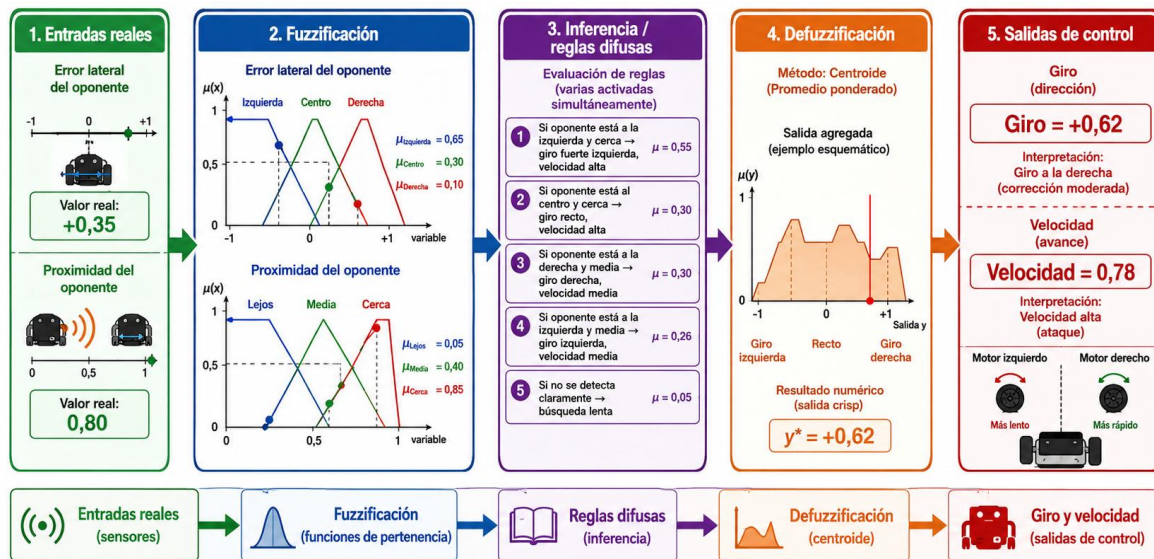


Figura 17.4: Proceso de fuzzificación, inferencia y defuzzificación.

17.8. Variables del controlador

El error lateral se define entre -100 y 100 . Un valor negativo representa oponente hacia la izquierda, cero representa oponente centrado y positivo representa oponente hacia la derecha.

La proximidad se define entre 0 y 100 . Cero representa oponente no detectado o lejano, mientras que 100 representa oponente muy cercano.

La salida de giro se define entre -100 y 100 . Valores negativos giran hacia la izquierda y valores positivos hacia la derecha. La velocidad se define entre 0 y 255 , compatible con PWM [7].

17.9. Base de reglas

Tabla 17.2: Reglas difusas propuestas para ataque de Minisumo.

Error lateral	Proximidad	Giro	Velocidad
Izquierda	Cerca	Izquierda fuerte	Alta
Izquierda	Media	Izquierda media	Media
Izquierda	Lejos	Izquierda suave	Baja
Centro	Cerca	Recto	Alta
Centro	Media	Recto	Media
Centro	Lejos	Recto	Baja
Derecha	Cerca	Derecha fuerte	Alta
Derecha	Media	Derecha media	Media
Derecha	Lejos	Derecha suave	Baja

En un Minisumo, la detección de borde debe estar por encima del control difuso. Si el robot detecta el borde del dohyo, debe ejecutar evasión, aunque el controlador indique atacar.

17.10. Diseño electrónico

Tabla 17.3: Asignación sugerida de pines para robot con control difuso.

Pin Arduino	Conexión	Función
A0	Sensor oponente izquierdo	Entrada analógica
A1	Sensor oponente centro	Entrada analógica
A2	Sensor oponente derecho	Entrada analógica
A3	Sensor borde izquierdo	Entrada analógica
A4	Sensor borde derecho	Entrada analógica
D3	PWMA TB6612FNG	PWM motor izquierdo
D4	AIN1 TB6612FNG	Dirección motor izquierdo
D5	AIN2 TB6612FNG	Dirección motor izquierdo
D6	PWMB TB6612FNG	PWM motor derecho
D7	BIN1 TB6612FNG	Dirección motor derecho
D8	BIN2 TB6612FNG	Dirección motor derecho
D9	STBY TB6612FNG	Habilitación del driver
D13	LED estado	Depuración

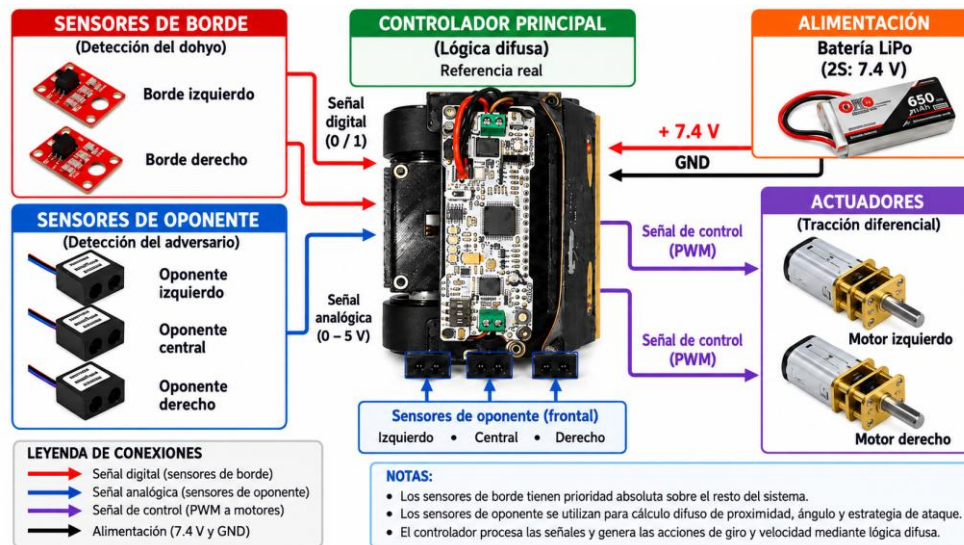


Figura 17.5: Diagrama de conexiones del robot con control difuso.

17.11. Código del proyecto

El siguiente programa implementa un controlador difuso básico sin librerías externas. Las funciones de pertenencia, reglas y defuzzificación se calculan directamente en el código.

```

const byte SENSOR_IZQ = A0;
const byte SENSOR_CEN = A1;
const byte SENSOR_DER = A2;
const byte BORDE_IZQ = A3;
const byte BORDE_DER = A4;

const byte PWMA = 3;
const byte AIN1 = 4;
const byte AIN2 = 5;
const byte PWMB = 6;
const byte BIN1 = 7;
const byte BIN2 = 8;
const byte STBY = 9;
const byte LED_ESTADO = 13;

const int UMBRAL_BORDE = 650;
const int SENSOR_MIN = 100;
const int SENSOR_MAX = 850;

int errorLateral = 0;
int proximidad = 0;
int salidaGiro = 0;
int salidaVelocidad = 0;

void motorIzquierdo(int velocidad) {
  velocidad = constrain(velocidad, -255, 255);
  if (velocidad > 0) {
    digitalWrite(AIN1, HIGH); digitalWrite(AIN2, LOW);
    analogWrite(PWMA, velocidad);
  } else if (velocidad < 0) {
    digitalWrite(AIN1, LOW); digitalWrite(AIN2, HIGH);
    analogWrite(PWMA, -velocidad);
  } else {
    digitalWrite(AIN1, LOW); digitalWrite(AIN2, LOW);
    analogWrite(PWMA, 0);
  }
}

void motorDerecho(int velocidad) {
  velocidad = constrain(velocidad, -255, 255);
  if (velocidad > 0) {
    digitalWrite(BIN1, HIGH); digitalWrite(BIN2, LOW);
    analogWrite(PWMB, velocidad);
  } else if (velocidad < 0) {
    digitalWrite(BIN1, LOW); digitalWrite(BIN2, HIGH);
    analogWrite(PWMB, -velocidad);
  } else {

```



```

    digitalWrite(BIN1, LOW); digitalWrite(BIN2, LOW);
    analogWrite(PWMB, 0);
}
}

void mover(int izquierda, int derecha) {
    motorIzquierdo(izquierda);
    motorDerecho(derecha);
}

float triangular(float x, float a, float b, float c) {
    if (x <= a || x >= c) return 0.0;
    if (x == b) return 1.0;
    if (x > a && x < b) return (x - a) / (b - a);
    return (c - x) / (c - b);
}

float hombroIzq(float x, float a, float b) {
    if (x <= a) return 1.0;
    if (x >= b) return 0.0;
    return (b - x) / (b - a);
}

float hombroDer(float x, float a, float b) {
    if (x <= a) return 0.0;
    if (x >= b) return 1.0;
    return (x - a) / (b - a);
}

int normalizarSensor(int lectura) {
    int valor = map(lectura, SENSOR_MIN, SENSOR_MAX, 0, 100);
    return constrain(valor, 0, 100);
}

void leerEntradas() {
    int izq = normalizarSensor(analogRead(SENSOR_IZQ));
    int cen = normalizarSensor(analogRead(SENSOR_CEN));
    int der = normalizarSensor(analogRead(SENSOR_DER));

    int suma = izq + cen + der;

    if (suma < 10) {
        errorLateral = 0;
        proximidad = 0;
        return;
    }
}

```

```

errorLateral = ((-100 * izq) + (0 * cen) + (100 * der)) / suma;
proximidad = max(cen, max(izq, der));
}

void controlDifuso() {
    float errIzq = hombroIzq(errorLateral, -80, 0);
    float errCen = triangular(errorLateral, -60, 0, 60);
    float errDer = hombroDer(errorLateral, 0, 80);

    float proxLejos = hombroIzq(proximidad, 20, 55);
    float proxMedio = triangular(proximidad, 30, 60, 90);
    float proxCerca = hombroDer(proximidad, 65, 95);

    float sumaPesos = 0;
    float sumaGiro = 0;
    float sumaVel = 0;
    float a;

    a = min(errIzq, proxCerca); sumaPesos += a; sumaGiro += a * (-100); sumaVel
    += a * 220;
    a = min(errIzq, proxMedio); sumaPesos += a; sumaGiro += a * (-70); sumaVel
    += a * 170;
    a = min(errIzq, proxLejos); sumaPesos += a; sumaGiro += a * (-45); sumaVel
    += a * 120;
    a = min(errCen, proxCerca); sumaPesos += a; sumaGiro += a * 0; sumaVel += a
    * 230;
    a = min(errCen, proxMedio); sumaPesos += a; sumaGiro += a * 0; sumaVel += a
    * 180;
    a = min(errCen, proxLejos); sumaPesos += a; sumaGiro += a * 0; sumaVel += a
    * 120;

    a = min(errDer, proxCerca); sumaPesos += a; sumaGiro += a * 100; sumaVel +=
    a * 220;
    a = min(errDer, proxMedio); sumaPesos += a; sumaGiro += a * 70; sumaVel += a
    * 170;
    a = min(errDer, proxLejos); sumaPesos += a; sumaGiro += a * 45; sumaVel += a
    * 120;

    if (sumaPesos <= 0.001) {
        salidaGiro = 70;
        salidaVelocidad = 90;
    } else {
        salidaGiro = sumaGiro / sumaPesos;
        salidaVelocidad = sumaVel / sumaPesos;
    }
}

```

```

bool detectarBordeIzq() {
    return analogRead(BORDE_IZQ) > UMBRAL_BORDE;
}

bool detectarBordeDer() {
    return analogRead(BORDE_DER) > UMBRAL_BORDE;
}

void evadirBorde() {
    bool bordeIzq = detectarBordeIzq();
    bool bordeDer = detectarBordeDer();

    digitalWrite(LED_ESTADO, HIGH);

    if (bordeIzq && bordeDer) {
        mover(-180, -180); delay(220);
        mover(170, -170); delay(250);
    } else if (bordeIzq) {
        mover(-160, -160); delay(180);
        mover(170, -120); delay(220);
    } else if (bordeDer) {
        mover(-160, -160); delay(180);
        mover(-120, 170); delay(220);
    }

    digitalWrite(LED_ESTADO, LOW);
}

void aplicarMovimientoDifuso() {
    int giroPWM = map(salidaGiro, -100, 100, -120, 120);

    int pwmIzq = salidaVelocidad + giroPWM;
    int pwmDer = salidaVelocidad - giroPWM;

    pwmIzq = constrain(pwmIzq, -255, 255);
    pwmDer = constrain(pwmDer, -255, 255);

    mover(pwmIzq, pwmDer);
}

void enviarDatos() {
    static unsigned long t = 0;
    if (millis() - t >= 250) {
        t = millis();
        Serial.print("Error:"); Serial.print(errorLateral);
        Serial.print(" Prox:"); Serial.print(proximidad);
        Serial.print(" Giro:"); Serial.print(salidaGiro);
    }
}

```

```

    Serial.print(" Vel:"); Serial.println(salidaVelocidad);
  }
}

void setup() {
  pinMode(PWMA, OUTPUT); pinMode(AIN1, OUTPUT); pinMode(AIN2, OUTPUT);
  pinMode(PWMB, OUTPUT); pinMode(BIN1, OUTPUT); pinMode(BIN2, OUTPUT);
  pinMode(STBY, OUTPUT); pinMode(LED_ESTADO, OUTPUT);
  digitalWrite(STBY, HIGH);
  Serial.begin(115200);
  Serial.println("Control difuso iniciado");
}

void loop() {
  if (detectarBordeIzq() || detectarBordeDer()) {
    evadirBorde();
    return;
  }

  leerEntradas();
  controlDifuso();
  aplicarMovimientoDifuso();
  enviarDatos();
}

```

El código asume que lecturas mayores indican mayor detección. Si los sensores entregan valores invertidos, se debe modificar la función `normalizarSensor()`.

17.12. Comparación con máquina de estados

El control difuso no reemplaza necesariamente a una máquina de estados. En muchos robots competitivos, ambas estrategias pueden combinarse. La máquina de estados organiza comportamientos generales, como esperar, buscar, atacar y evadir. El control difuso suaviza decisiones dentro de un estado, por ejemplo, durante el ataque.

Tabla 17.4: Comparación entre control por condiciones y control difuso.

Criterio	Condiciones if	Control difuso
Implementación inicial	Muy simple	Moderada
Transiciones	Bruscas	Suaves
Ajuste fino	Limitado	Flexible
Comprensión inicial	Fácil	Requiere práctica
Robustez ante ruido	Depende del filtrado	Puede ser mayor
Costo computacional	Bajo	moderado

17.13. Procedimiento de prueba

- 1. Verificar que los sensores de oponente aumenten su lectura al acercar un objeto.
- 2. Mover un objeto de izquierda a derecha y observar que el error lateral pase de negativo a positivo.
- 3. Elevar el robot y observar en el monitor serial los valores de giro y velocidad.
- 4. Probar movimientos en mesa con velocidad limitada antes de usar el dohyo.
- 5. Probar en el dohyo la búsqueda, ataque y evasión de borde.

Tabla 17.5: Registro sugerido de pruebas del controlador difuso.

Prueba	Condición	Resultado esperado	Resultado obtenido
Objeto izquierda	Cerca	Giro fuerte izquierda	
Objeto centro	Cerca	Ataque recto rápido	
Objeto derecha	Medio	Giro medio derecha	
Sin objeto	Ninguna detección	Búsqueda giratoria	
Borde detectado	Sensor de borde activo	Evasión inmediata	

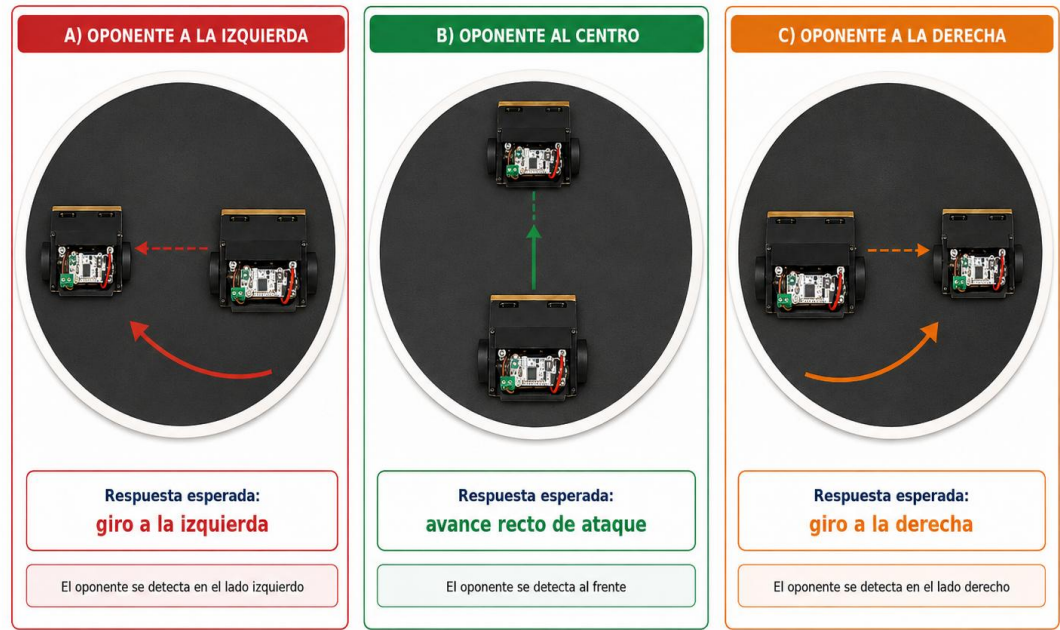


Figura 17.6: Validación experimental del controlador difuso.

17.14. Ajuste del controlador difuso

Para ajustar el controlador se pueden modificar los rangos de pertenencia, los valores de salida de giro, los valores de velocidad y las reglas activadas. Si el robot gira demasiado agresivo, reducir los valores de giro. Si ataca lento, aumentar velocidad en reglas de proximidad cercana. Si busca demasiado rápido y pierde al oponente, reducir velocidad de búsqueda [4].

17.15. Errores frecuentes

- Definir funciones de pertenencia que no se traslapan.
- Usar reglas contradictorias sin entender su efecto.
- No normalizar sensores antes de aplicar lógica difusa.
- Olvidar la prioridad de borde en Minisumo.
- Usar velocidad alta desde las primeras pruebas.
- Interpretar mal el signo del error lateral.
- Creer que el control difuso corrige problemas mecánicos graves.

17.16. Proyecto de extensión

Como extensión, se propone implementar perfiles difusos seleccionables:

- **Perfil conservador:** giros suaves y velocidad media.
- **Perfil agresivo:** ataque rápido y giros fuertes.
- **Perfil defensivo:** búsqueda lenta y evasión prioritaria.

Otra extensión consiste en aplicar control difuso a un seguidor de línea, usando como entrada el error de línea y su variación, y como salida la corrección de motores.



Figura 17.7: Extensión con perfiles difusos seleccionables.

Actividades de aprendizaje

1. Grafique las funciones de pertenencia de error lateral y proximidad.
2. Modifique los valores de salida de giro y observe el comportamiento del robot.
3. Agregue una regla para ataque extremadamente cercano.
4. Compare el desempeño usando control por if y control difuso.

5. Diseñe un perfil difuso agresivo y uno conservador.
6. Proponga una aplicación del control difuso para un seguidor de línea.

Preguntas de reflexión

1. ¿Qué ventaja ofrece el control difuso frente a condiciones rígidas?
2. ¿Por qué las funciones de pertenencia deben traslaparse?
3. ¿Qué ocurre si dos reglas se activan al mismo tiempo?
4. ¿Por qué la detección de borde debe tener prioridad sobre el ataque?
5. ¿Cómo se podría usar control difuso en un robot seguidor de línea?
6. ¿Qué limitaciones tiene el control difuso en robots competitivos reales?

Referencias

- [1] L. A. Zadeh, “Fuzzy sets”, *Information and Control*, vol. 8, no. 3, pp. 338-353, 1965.
- [2] T. J. Ross, *Fuzzy Logic with Engineering Applications*, 4th ed. Hoboken, NJ, USA: Wiley, 2016.
- [3] K. M. Passino and S. Yurkovich, *Fuzzy Control*. Menlo Park, CA, USA: AddisonWesley, 1998.
- [4] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ, USA: Princeton University Press, 2008.
- [5] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed. Cham, Switzerland: Springer, 2016.
- [6] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.
- [7] Toshiba Electronic Devices & Storage Corporation, “TB6612FNG Brush Motor Driver IC”, Toshiba. [En línea]. Disponible en: <https://toshiba.semicon-storage.com/ap-en/semiconductor/product/motor-driver-ics/brushed-dc-motor-driver-ics/detail.TB6612FNG.html>. [Accedido: 24-jun-2026].

Capítulo 18. Registro de Datos con microSD para Robots Competitivos

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender la importancia del registro de datos en robots competitivos y sistemas embebidos.
- Diseñar un data logger con Arduino y módulo microSD.
- Guardar datos experimentales en formato CSV.
- Registrar variables como voltaje de batería, corriente, sensores, PWM, estado del robot y eventos.
- Implementar nombres de archivo automáticos para no sobrescribir pruebas anteriores.
- Analizar datos registrados en una hoja de cálculo.
- Aplicar el registro de datos a seguidores de línea, Minisumo, robots exploradores y bancos de prueba.

18.1. Introducción

Con una estación externa de cronometraje para medir el desempeño de seguidores de línea. Ese sistema permite comparar perfiles de control a partir de tiempos de vuelta. Sin embargo, el tiempo final no siempre explica por qué un robot mejora o falla. Para comprender el comportamiento interno del robot se necesita registrar datos durante la prueba.

Un robot competitivo toma decisiones en fracciones de segundo. Puede detectar una línea, perderla, corregir con los motores, activar una evasión de borde, sufrir caída de voltaje o saturar el PWM. Muchas de estas situaciones no se observan claramente a simple vista. Por ello, un sistema de registro de datos resulta fundamental.

Este capítulo desarrolla un data logger embebido con microSD. El sistema guarda información en formato CSV, de manera que pueda abrirse luego en una hoja de cálculo. La solución es aplicable a seguidores de línea, Minisumo, robots exploradores, sistemas de telemetría, bancos de motores y plataformas educativas.

El objetivo no es registrar todos los datos posibles, sino seleccionar variables útiles para responder preguntas de ingeniería:

- ¿En qué momento cae el voltaje?
- ¿Cuándo se pierde la línea?
- ¿Qué PWM se aplicó en cada curva?
- ¿Cuántas veces se activó una evasión?

- ¿Cuál fue el error máximo?
- ¿Qué perfil de control fue más estable?

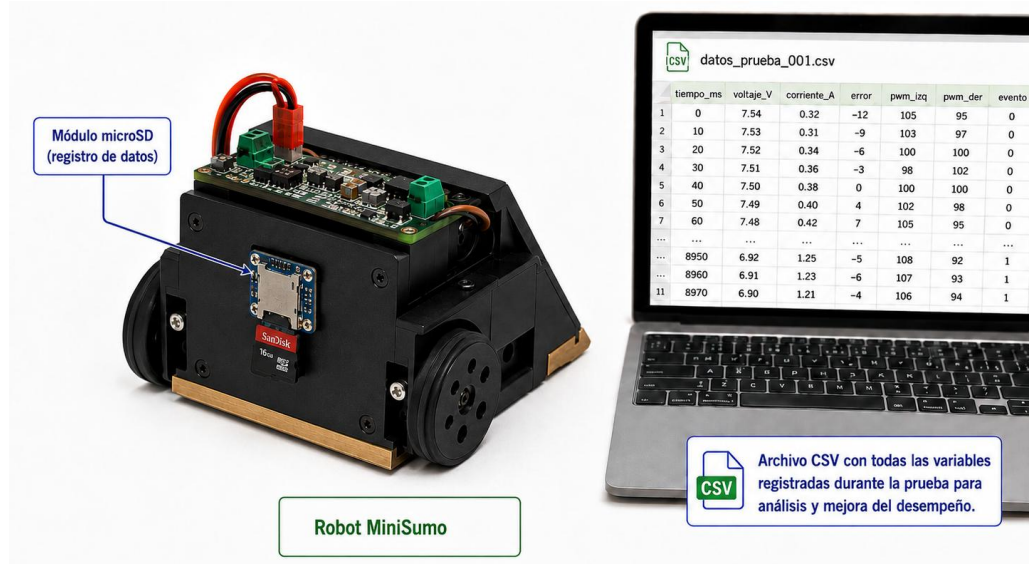


Figura 18.1: Registro de datos con microSD en robots competitivos.

18.2. Planteamiento del problema

Durante la puesta a punto de un robot, el equipo suele modificar parámetros y observar el resultado. Este método es rápido, pero puede ser subjetivo. A veces el robot parece más veloz, pero en realidad oscila más. O parece más estable, pero pierde tiempo en curvas. También puede fallar por una causa eléctrica y no por el algoritmo [6].

Sin registro de datos, el análisis queda limitado a observación visual y memoria del operador. Esto genera problemas:

- No se sabe exactamente cuándo ocurrió la falla.
- No se puede comparar una prueba con otra de forma objetiva.
- No se detectan caídas de batería durante aceleraciones.
- No se registra el comportamiento del PWM.
- No se puede reconstruir la secuencia de eventos.
- Se repiten errores de ajuste por falta de evidencia.

El data logger permite guardar la historia interna del robot durante una prueba.

La telemetría muestra lo que ocurre en tiempo real. El registro en microSD permite estudiar después lo que ocurrió durante toda la prueba.

18.3. Descripción general del proyecto

El proyecto consiste en conectar un módulo microSD a un Arduino Nano o compatible mediante comunicación SPI. El programa crea un archivo CSV y registra periódicamente variables del robot [1, 7].

En una versión básica se registran:

- Tiempo en milisegundos.
- Voltaje de batería.
- Corriente estimada [5].
- Estado del robot.
- Error de control.
- PWM izquierdo.
- PWM derecho.
- Eventos o fallas.

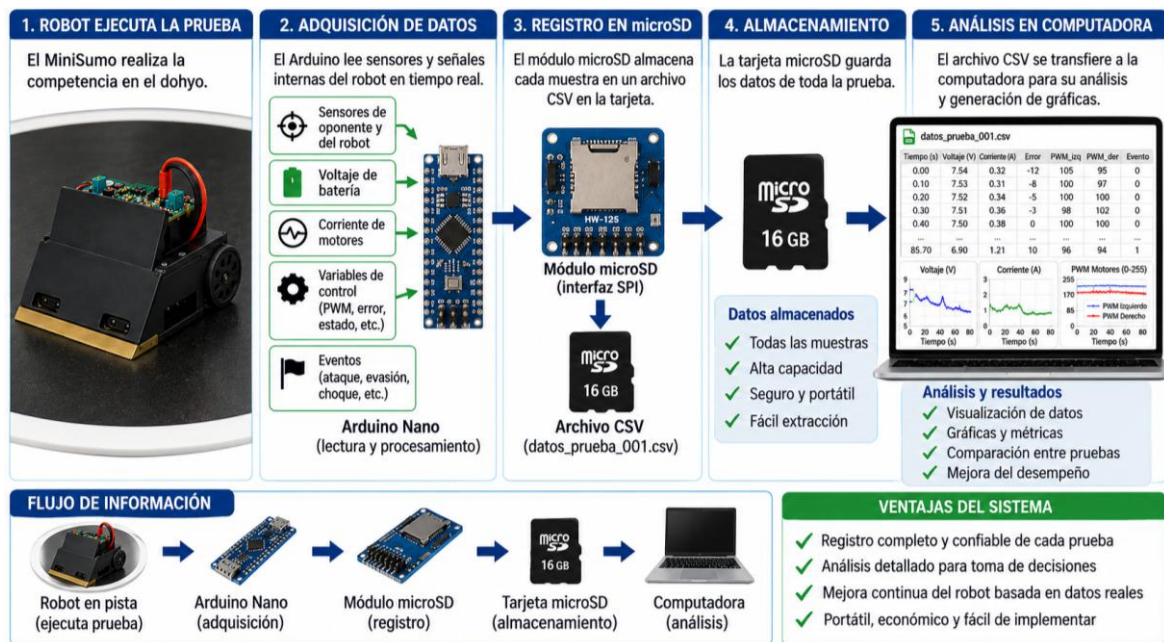


Figura 18.2: Arquitectura general del sistema de registro de datos con microSD.

18.4. Aplicación STEM

Este proyecto integra las áreas STEM de la siguiente manera:

- **Ciencia:** medición, adquisición de datos, análisis experimental y comportamiento eléctrico.
- **Tecnología:** microSD, comunicación SPI, sensores, microcontroladores y archivos CSV.

- **Ingeniería:** diagnóstico de fallas y validación de robots reales.
- **Matemáticas:** promedios, máximos, mínimos, correlación entre variables y comparación de pruebas.

18.5. Objetivos del proyecto

18.5.1. Objetivo general

Diseñar e implementar un sistema de registro de datos con microSD para robots competitivos, capaz de guardar variables experimentales en formato CSV para análisis posterior.

18.5.2. Objetivos específicos

1. Configurar un módulo microSD mediante comunicación SPI.
2. Crear archivos CSV con encabezados descriptivos.
3. Registrar datos periódicamente sin bloquear el control principal.
4. Medir voltaje de batería mediante divisor resistivo.
5. Registrar variables de control como error, PWM y estado.
6. Generar nombres de archivo automáticos.
7. Analizar los datos en una hoja de cálculo.

18.6. Materiales y herramientas

1  Arduino Nano (control y adquisición)	2  Módulo microSD (interfaz SPI)	3  Tarjeta microSD (16 GB o superior)	4  Divisor resistivo (medición de voltaje)
5  Sensor de corriente (ACS712 o similar)	6  LED (indicación de registro)	7  Pulsador (marcado de eventos)	8  Robot de prueba (MiniSumo)
9  Computadora (almacenamiento y análisis de datos) FUNCIÓN GENERAL DE LOS MATERIALES ✓ Adquirir variables relevantes del robot en tiempo real. ✓ Guardar las muestras en la tarjeta microSD. ✓ Permitir el análisis y mejora del desempeño del robot.			

Figura 18.3: Materiales utilizados para registro de datos con microSD.

Tabla 18.1: Materiales principales para el data logger con microSD.

Elemento	Cantidad	Función
Arduino Nano o compatible	1	Control y registro
Módulo microSD	1	Almacenamiento de datos
Tarjeta microSD	1	Memoria de registro
Divisor resistivo	1	Medición de voltaje de batería
Sensor de corriente ACS712 o similar	1	Medición de consumo
LED indicador	1	Estado de grabación
Botón pulsador	1	Inicio o marca de evento
Robot de prueba	1	Plataforma experimental
Computadora	1	Análisis de archivos CSV

18.7. Fundamento teórico

18.7.1. Comunicación SPI

Los módulos microSD se comunican generalmente mediante SPI. En Arduino Nano, los pines SPI son [2]:

- D11: MOSI.
- D12: MISO.
- D13: SCK.
- D10: CS o selección de tarjeta.

El pin CS puede variar, pero D10 suele utilizarse en Arduino Nano.

18.7.2. Formato CSV

CSV significa *Comma-Separated Values*. Es un formato de texto donde cada línea representa una muestra y cada variable se separa por comas. Ejemplo:

```
tiempo_ms,voltaje,corriente,estado,error,pwm_izq,pwm_der,eventos
0,12.45,0.35,ESPERA,0,0,0,0
100,12.40,1.20,AVANZAR,120,145,135,0
200,12.18,2.80,CURVA,2300,90,180,0
```

Este archivo puede abrirse en Excel, LibreOffice Calc, Google Sheets o software de análisis.

18.7.3. Frecuencia de muestreo

La frecuencia de muestreo define cuántas veces por segundo se guardan datos. Si se registra cada 100 ms, se obtienen 10 muestras por segundo:

$$f_s = \frac{1}{T_s}$$

Si $T_s = 0.1 \text{ s}$, entonces:

$$f_s = 10 \text{ Hz}$$

Una frecuencia muy baja puede perder eventos rápidos. Una frecuencia muy alta puede afectar el control del robot o saturar escritura en microSD.

18.7.4. Registro no bloqueante

El registro de datos no debe detener el control principal. Por ello se recomienda usar `millis()` en lugar de `delay()` [3, 8].

$$\text{Si } \text{millis}() - t_{\text{anterior}} \geq T_s \Rightarrow \text{guardamuestra}$$

18.7.5. Medición de voltaje

Para medir batería se utiliza un divisor resistivo [4]:

$$V_{ADC} = V_{BAT} \left(\frac{R_2}{R_1 + R_2} \right)$$

Por tanto:

$$V_{BAT} = V_{ADC} \left(\frac{R_1 + R_2}{R_2} \right)$$

Nunca conecte directamente una batería LiPo de varias celdas a una entrada analógica. Siempre debe usarse divisor resistivo y verificar que el voltaje al pin no supere el máximo permitido por el microcontrolador.

18.8. Diseño electrónico

Tabla 18.2: Asignación sugerida de pines para el data logger.

Pin Arduino	Conexión	Función
D10	CS microSD	Selección de tarjeta
D11	MOSI microSD	Datos hacia microSD
D12	MISO microSD	Datos desde microSD
D13	SCK microSD	Reloj SPI
A0	Divisor de voltaje	Medición de batería
A1	Sensor de corriente	Medición de consumo
D2	Botón evento	Marca manual de evento
D7	LED registro	Indica escritura activa
5 V / 3.3 v	Módulo microSD	Alimentación según módulo
GND	Tierra común	Referencia eléctrica

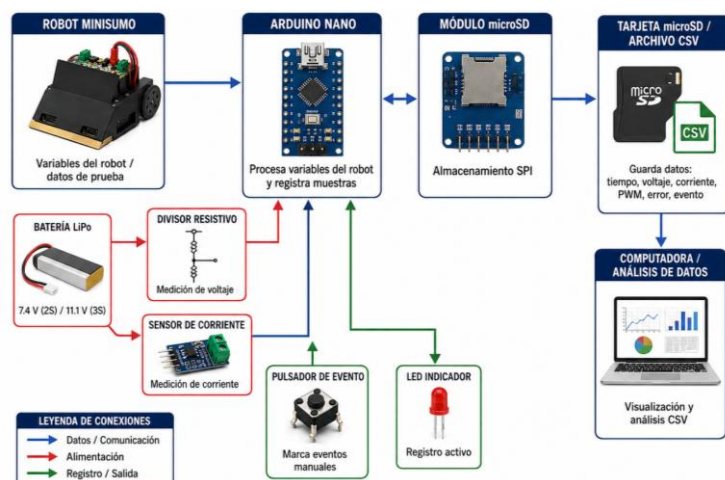


Figura 18.4: Diagrama de conexiones del data logger con microSD.

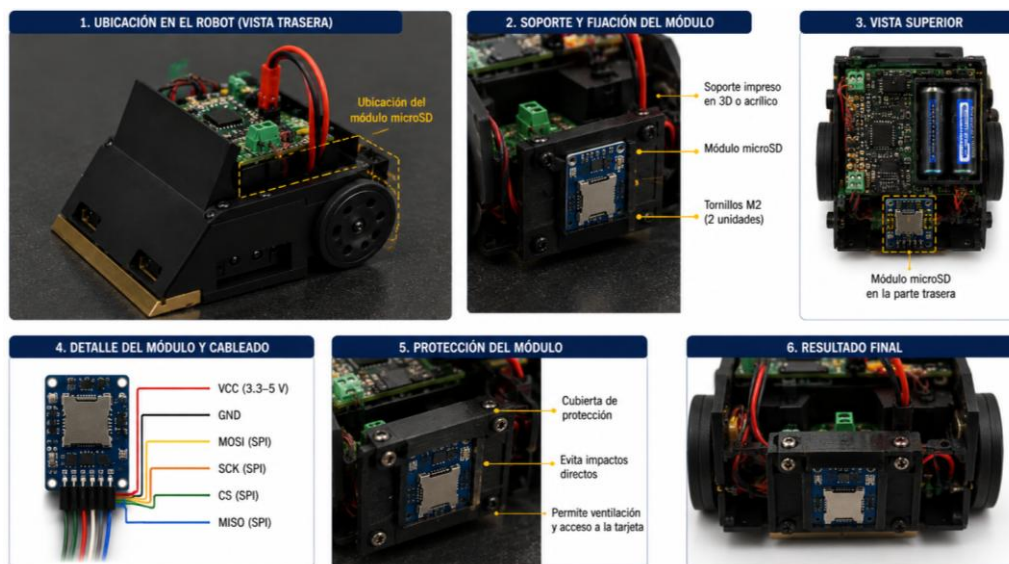


Figura 18.5: Montaje físico del módulo microSD en el robot.

18.9. Selección de variables

No siempre conviene guardar todas las variables. Un buen registro debe ser útil y manejable.

Tabla 18.3: Variables recomendadas según tipo de robot.

Robot	Variables principales	Objetivo de análisis
Seguidor de línea	error, derivada, PWM, pérdida de línea	Ajustar PID y curvas
Minisumo	sensores oponente, borde, estado, PWM	Evaluar ataque y evasión
Explorador	distancia, estado, giro, batería	Diagnóstico de navegación
Robot con encoders	RPM, distancia, orientación	Validar odometría
Robot de combate	corriente, voltaje, eventos, temperatura	Protección y desempeño

18.10. Código del proyecto

El siguiente programa crea un archivo CSV en la microSD y registra variables representativas. Puede integrarse con otros capítulos reemplazando las variables simuladas por variables reales del robot.

```
#include <SPI.h>
#include <SD.h>

// =====
// PINES
// =====
const byte PIN_CS_SD = 10;
const byte PIN_VBAT = A0;
const byte PIN_CORRIENTE = A1;
const byte PIN_EVENTO = 2;
const byte PIN_LED = 7;

// =====
// PARAMETROS DE MEDICION
// =====
const float VREF = 5.0;
const float ADC_MAX = 1023.0;

const float R1 = 100000.0;
const float R2 = 10000.0;

// Sensor corriente referencial
float offsetCorriente = 2.5;
float sensibilidadCorriente = 0.100; // V/A, ajustar segun sensor

// =====
// PARAMETROS DE REGISTRO
// =====
const unsigned long INTERVALO_LOG = 100; // ms

File archivo;
char nombreArchivo[13];

unsigned long tiempoAnteriorLog = 0;
unsigned long contadorMuestras = 0;

int contadorEventos = 0;

// Variables representativas del robot
int estadoRobot = 0;
int errorControl = 0;
```

```

int pwmIzq = 0;
int pwmDer = 0;

// =====
// CREAM NOMBRE AUTOMATICO
// =====
bool crearNombreArchivo()
{
    for (int i = 0; i < 100; i++)
    {
        sprintf(nombreArchivo, "LOG %02d.CSV", i);

        if (!SD.exists(nombreArchivo))
        {
            return true;
        }
    }

    return false;
}

// =====
// LECTURAS ANALOGICAS
// =====
float leerVoltajeADC(byte pin)
{
    int adc = analogRead(pin);
    return (adc * VREF) / ADC_MAX;
}

float leerBateria()
{
    float vADC = leerVoltajeADC(PIN_VBAT);
    float vBat = vADC * ((R1 + R2) / R2);
    return vBat;
}

float leerCorriente()
{
    float v = leerVoltajeADC(PIN_CORRIENTE);
    float i = (v - offsetCorriente) / sensibilidadCorriente;

    if (i < 0)
    {
        i = -i;
    }
}

```



```

    return i;
}

// =====
// VARIABLES DE EJEMPLO
// Reemplazar por variables reales del robot
// =====
void actualizarVariablesRobot()
{
    // Ejemplo educativo:
    // En un seguidor real, errorControl vendria del sensor de linea.
    // pwmIzq y pwmDer vendrian del control de motores.

    unsigned long t = millis();

    estadoRobot = (t / 1000) % 4;
    errorControl = analogRead(A2) - 512;
    pwmIzq = map(analogRead(A3), 0, 1023, 0, 255);
    pwmDer = map(analogRead(A4), 0, 1023, 0, 255);
}

// =====
// EVENTOS
// =====
void revisarEvento()
{
    static bool estadoAnterior = HIGH;
    bool estado = digitalRead(PIN_EVENTO);

    if (estadoAnterior == HIGH && estado == LOW)
    {
        contadorEventos++;
    }

    estadoAnterior = estado;
}

// =====
// INICIAR ARCHIVO
// =====
bool iniciarLogger()
{
    if (!SD.begin(PIN_CS_SD))
    {
        return false;
    }
}

```

```

if (!crearNombreArchivo())
{
    return false;
}

archivo = SD.open(nombreArchivo, FILE_WRITE);

if (!archivo)
{
    return false;
}

archivo.println("muestra,tiempo_ms,voltaje,corriente,estado,error,pwm_izq
,pwm_der,eventos");
archivo.flush();

return true;
}

// =====
// GUARDAR MUESTRA
// =====
void guardarMuestra()
{
    float voltaje = leerBateria();
    float corriente = leerCorriente();

    archivo.print(contadorMuestras);
    archivo.print(",");
    archivo.print(millis());
    archivo.print(",");
    archivo.print(voltaje, 2);
    archivo.print(",");
    archivo.print(corriente, 2);
    archivo.print(",");
    archivo.print(estadosRobot);
    archivo.print(",");
    archivo.print(errorControl);
    archivo.print(",");
    archivo.print(pwmIzq);
    archivo.print(",");
    archivo.print(pwmDer);
    archivo.print(",");
    archivo.println(contadorEventos);

    contadorMuestras++;
}

```

```

// flush periodico para reducir riesgo de perdida de datos
if (contadorMuestras % 10 == 0)
{
    archivo.flush();
}
}

// =====
// SETUP Y LOOP
// =====

void setup()
{
    pinMode(PIN_EVENTO, INPUT_PULLUP);
    pinMode(PIN_LED, OUTPUT);

    Serial.begin(115200);

    Serial.println("Iniciando data logger...");

    if (iniciarLogger())
    {
        Serial.print("Archivo creado: ");
        Serial.println(nombreArchivo);
        digitalWrite(PIN_LED, HIGH);
    }
    else
    {
        Serial.println("Error al iniciar microSD");
        digitalWrite(PIN_LED, LOW);

        while (true)
        {
            digitalWrite(PIN_LED, !digitalRead(PIN_LED));
            delay(250);
        }
    }
}

void loop()
{
    revisarEvento();
    actualizarVariablesRobot();

    unsigned long ahora = millis();

    if (ahora - tiempoAnteriorLog >= INTERVALO_LOG)
    {

```

```

tiempoAnteriorLog = ahora;
guardarMuestra();

Serial.print("Muestra guardada: ");
Serial.println(contadorMuestras);
}
}

```

Para robots de competencia, el data logger debe probarse primero con el robot elevado o en banco. La escritura en microSD no debe interferir con el control principal ni provocar reinicios por consumo o ruido eléctrico.

18.11. Integración con un seguidor de línea

En un seguidor de línea, las variables más importantes son:

- Tiempo.
- Error de línea.
- Derivada del error.
- PWM izquierdo.
- PWM derecho.
- Velocidad base.
- Perfil activo.
- Pérdidas de línea.

El registro permite analizar si una curva cerrada produce saturación de PWM o si la velocidad adaptativa reduce correctamente en curvas.

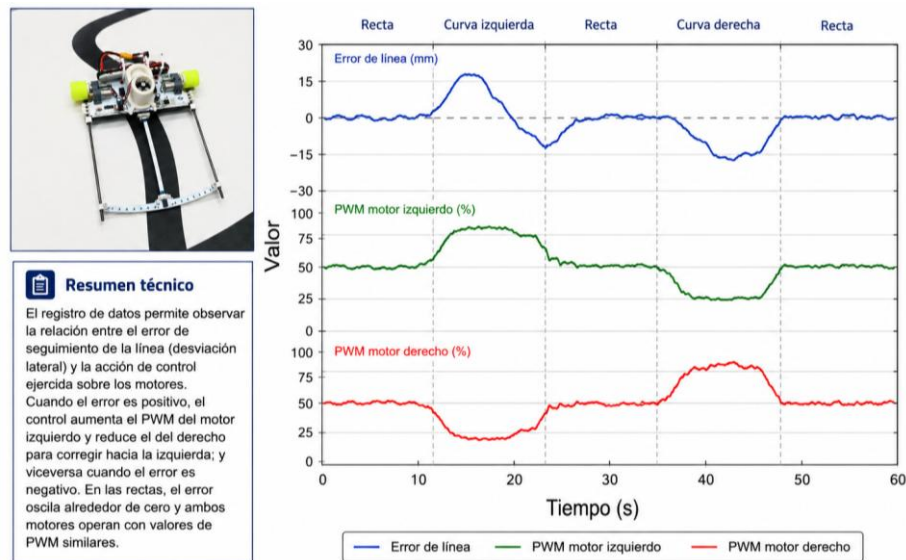


Figura 18.6: Análisis de datos registrados en un seguidor de línea.

18.12. Integración con un Minisumo

En un Minisumo, el data logger puede registrar:

- Estado: buscar, atacar, evadir, recuperar.
- Sensores de oponente.
- Sensores de borde.
- PWM de motores.
- Voltaje de batería.
- Eventos de evasión.

Esto permite estudiar si el robot pierde tiempo buscando, si evade demasiado tarde o si la batería cae durante empujes prolongados.

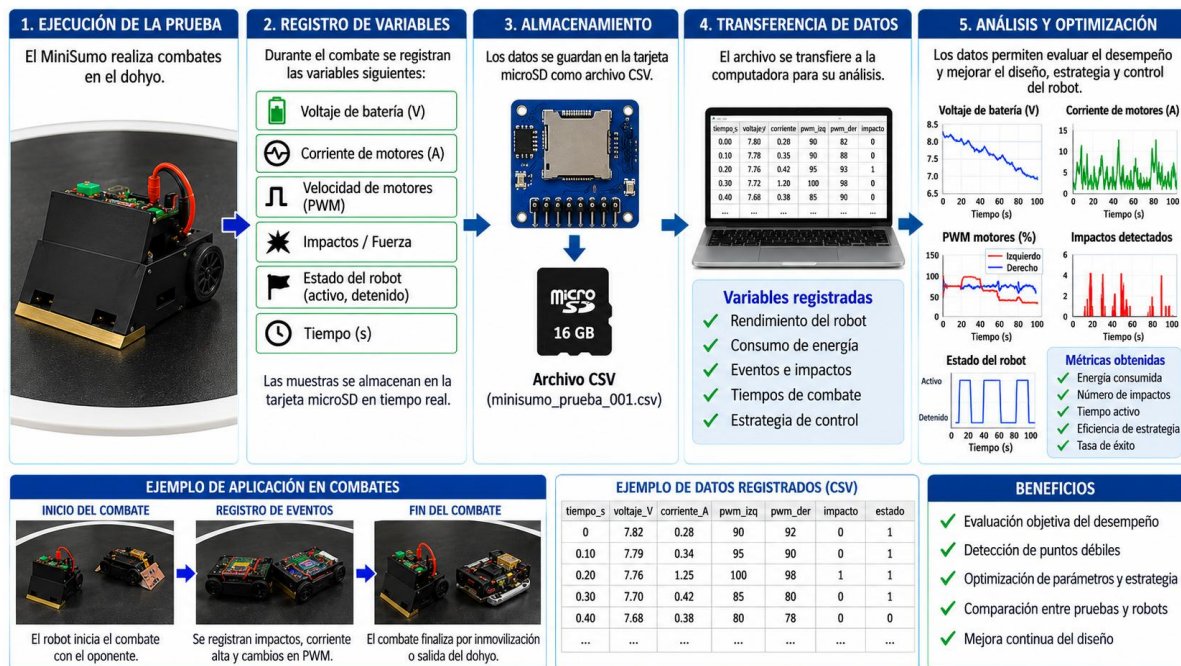


Figura 18.7: Análisis de datos registrados en un seguidor de línea.

18.13. Análisis en hoja de cálculo

Una vez finalizada la prueba:

1. Retirar la microSD.
2. Copiar el archivo CSV a la computadora.
3. Abrirlo en una hoja de cálculo.
4. Graficar variables principales.
5. Identificar máximos, mínimos y eventos.
6. Comparar con otros archivos de prueba.

Tabla 18.4: Preguntas de análisis a partir del archivo CSV.

Pregunta	Variable útil
¿Dónde se perdió la línea?	error, eventos, tiempo
¿Hubo saturación de motores?	PWM izquierdo y derecho
¿La batería cayó demasiado?	voltaje
¿El robot consumió mucho en curvas?	corriente, PWM, error
¿El perfil rápido fue estable?	error máximo, eventos, tiempo
¿El robot evadió muchas veces?	error máximo, eventos, tiempo

18.14. Frecuencia de registro recomendada

Tabla 18.5: Frecuencias de registro sugeridas según aplicación.

Aplicación	Intervalo	Comentario
Telemetría lenta de batería	500 – 1000 ms	Suficiente para descarga general
Minisumo	50 – 100 ms	Captura eventos de ataque y evasión
Seguidor de línea	20 – 50 ms	Requiere más detalle en curvas
Banco e motores	20 – 100 ms	Depende de la respuesta del motor
Robot explorador	100 – 250 ms	Navegación lenta

18.15. Buenas prácticas

- Usar tarjetas microSD confiables y formateadas.
- Sujetar bien el módulo para evitar desconexión por vibración.
- Usar cables cortos para SPI.
- Evitar compartir alimentación ruidosa con motores.
- Llamar a flush() periódicamente.
- No retirar la tarjeta mientras se escribe.
- Nombrar archivos de forma automática para no sobrescribir pruebas.
- Documentar qué configuración se usó en cada archivo.

18.16. Errores frecuentes

Durante el desarrollo pueden presentarse los siguientes errores:

- El módulo microSD no inicia por alimentación incorrecta.
- Se conectan mal los pines SPI.
- La tarjeta no está formateada correctamente.
- El archivo se corrompe por retirar la tarjeta durante escritura.
- El registro es demasiado frecuente y afecta el control.
- Se guardan datos sin encabezados, dificultando el análisis.
- No se documenta qué prueba corresponde a cada archivo.
- El módulo se desconecta por vibración.

18.17. Proyecto de extensión

Como extensión, se propone combinar microSD con dashboard inalámbrico. Durante la prueba, el ESP32 muestra datos en vivo; al mismo tiempo, la microSD guarda el historial completo.

Otra extensión consiste en añadir un reloj en tiempo real, como DS3231, para registrar fecha y hora de cada prueba. Esto permite llevar un historial ordenado de entrenamientos y competencias.

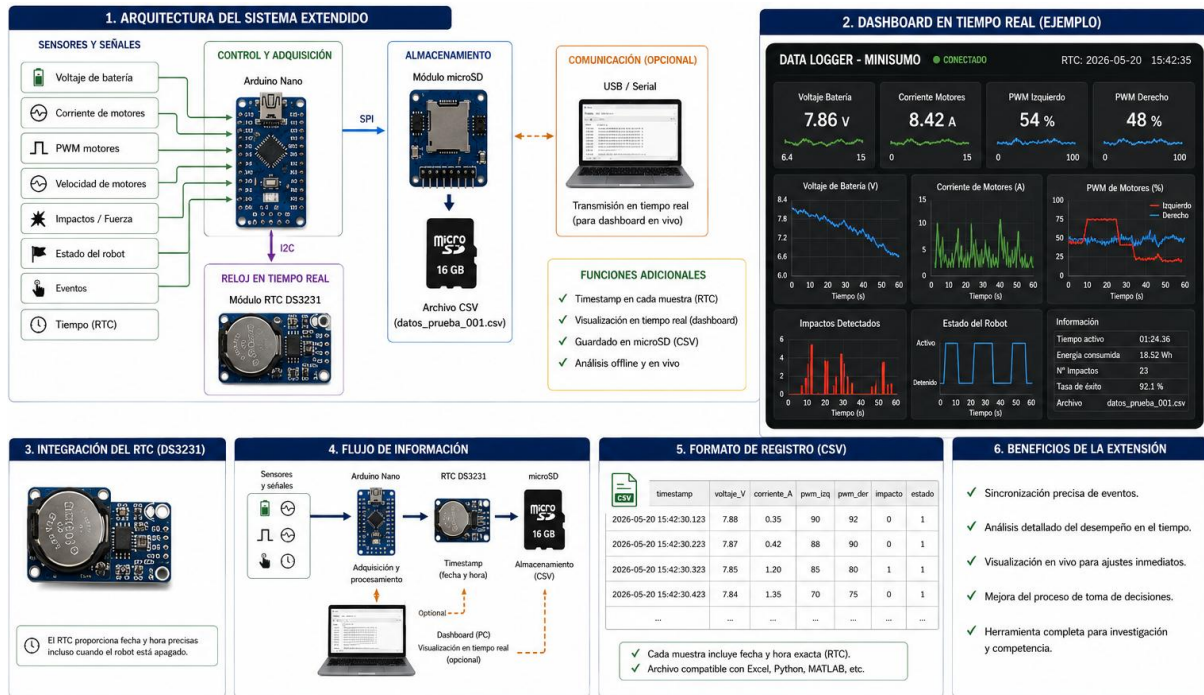


Figura 18.8: Extensión del data logger con dashboard y reloj en tiempo real.

Actividades de aprendizaje

1. Monte el módulo microSD y cree un archivo CSV de prueba.
2. Registre voltaje de batería durante una prueba de 2 minutos.
3. Integre el registro de error y PWM en un seguidor de línea.
4. Compare dos perfiles de control usando archivos CSV.
5. Grafique error de línea y PWM en una hoja de cálculo.
6. Proponga una versión del data logger para Minisumo.

Preguntas de reflexión

1. ¿Qué diferencia existe entre telemetría en vivo y registro en microSD?
2. ¿Por qué el formato CSV es útil para análisis experimental?
3. ¿Qué variables registraría en un robot seguidor de línea durante una vuelta completa?
4. ¿Qué riesgos existen si se escribe en microSD con demasiada frecuencia?

5. ¿Cómo puede el data logger ayudar a detectar fallas de batería?
6. ¿Cómo integraría este sistema con cronometraje por sectores?

Referencias

- [1] Arduino, “SD Library”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/libraries/sd/>. [Accedido: 25-jun-2026].
- [2] Arduino, “SPI Library”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/learn/communication/spi>. [Accedido: 25-jun-2026].
- [3] Arduino, “millis()”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/language-reference/en/functions/time/millis/>. [Accedido: 25-jun-2026].
- [4] Arduino, “analogRead()”, Arduino Documentation. [En línea]. Disponible en: <https://docs.arduino.cc/language-reference/en/functions/analog-io/analogRead/>. [Accedido: 25-jun-2026].
- [5] Allegro MicroSystems, “ACS712 Datasheet”, Allegro MicroSystems. [En línea]. Disponible en: <https://www.allegromicro.com/-/media/files/datasheets/acs712-datasheet.ashx>. [Accedido: 25-jun-2026].
- [6] D. C. Montgomery, *Design and Analysis of Experiments*, 10th ed. Hoboken, NJ, USA: Wiley, 2019.
- [7] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2017.
- [8] M. Barr and A. Massa, *Programming Embedded Systems: With C and GNU Development Tools*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2006.

Capítulo 19. Análisis de Datos y Optimización Experimental con Python

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender la importancia del análisis de datos en el ajuste de robots competitivos.
- Importar archivos CSV generados por microSD, telemetría o cronometraje.
- Utilizar Python, pandas y matplotlib para procesar datos experimentales.
- Calcular métricas de desempeño como mejor tiempo, promedio, error máximo, saturación de PWM y caída de voltaje.
- Comparar perfiles de control PID, PID-Difuso y configuraciones mecánicas.
- Generar gráficas para interpretar el comportamiento del robot durante una prueba.
- Tomar decisiones de ajuste basadas en evidencia y no solo en observación visual.

19.1. Introducción

En el Capítulo 18 se desarrolló un sistema de registro de datos con microSD para guardar información interna del robot durante una prueba. Ese archivo CSV contiene una historia detallada del comportamiento del sistema: tiempo, voltaje, corriente, estado, error, PWM, eventos y otras variables. Sin embargo, los datos por sí solos no producen mejoras. Es necesario analizarlos.

En robótica competitiva, la diferencia entre un robot rápido y un robot ganador muchas veces está en el proceso de ajuste. Un equipo puede cambiar K_p , K_d , velocidad, altura de sensores, llantas, batería o estrategia, pero si no compara los resultados de forma ordenada, puede tomar decisiones equivocadas. Un ajuste puede parecer mejor en video, pero generar más oscilación, mayor consumo o más riesgo de pérdida de línea.

Este capítulo introduce una metodología de análisis experimental con Python. Se utilizarán archivos CSV generados por los capítulos anteriores para calcular métricas, graficar señales y comparar perfiles de control. El objetivo no es convertir al lector en científico de datos, sino darle herramientas prácticas para responder preguntas concretas [1, 4, 8]:

- ¿Qué perfil fue más rápido?
- ¿Qué perfil fue más estable?
- ¿En qué momento cayó el voltaje?
- ¿Dónde se saturó el PWM?
- ¿Qué curva produjo mayor error?
- ¿Qué ajuste debería probarse después?

El capítulo está orientado a seguidores de línea, Minisumo, robots exploradores y bancos de motores. También puede aplicarse a robots del Club de Robótica UTE, como Shippu, Raiko, Kurogane, Kitsune u otras plataformas experimentales.

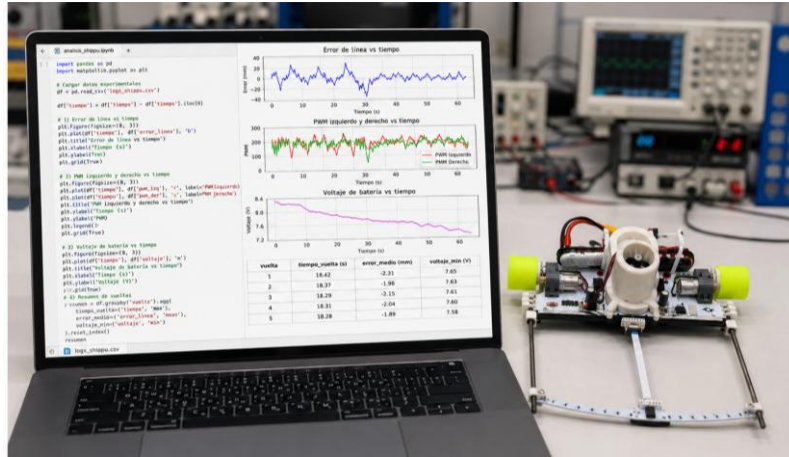


Figura 19.1: Análisis de datos experimentales de robots competitivos con Python.

19.2. Planteamiento del problema

Durante una jornada de pruebas, un equipo puede generar varios archivos:

- LOG00.CSV: perfil PID base.
- LOG01.CSV: aumento de velocidad.
- LOG02.CSV: aumento de K_d .
- LOG03.CSV: perfil PID-Difuso.
- LOG04.CSV: cambio de llantas.

Si solo se observa el robot, es difícil saber qué cambió realmente. Por ejemplo, una configuración puede lograr mejor tiempo, pero con mayor error máximo. Otra puede tener menor error, pero ser más lenta. Otra puede consumir más corriente y hacer caer el voltaje en curvas.

El problema central del capítulo es convertir datos en información útil para el ajuste del robot.

Registrar datos es solo la primera parte. La mejora real aparece cuando esos datos se transforman en conclusiones y decisiones de ajuste.

19.3. Descripción general del proyecto

El flujo de trabajo propuesto es:

1. Ejecutar pruebas del robot.
2. Guardar datos en microSD o recibirlos por Serial.

3. Copiar archivos CSV a la computadora.
4. Analizar los archivos con Python.
5. Calcular métricas de desempeño.
6. Generar gráficas.
7. Comparar configuraciones.
8. Decidir el siguiente ajuste.

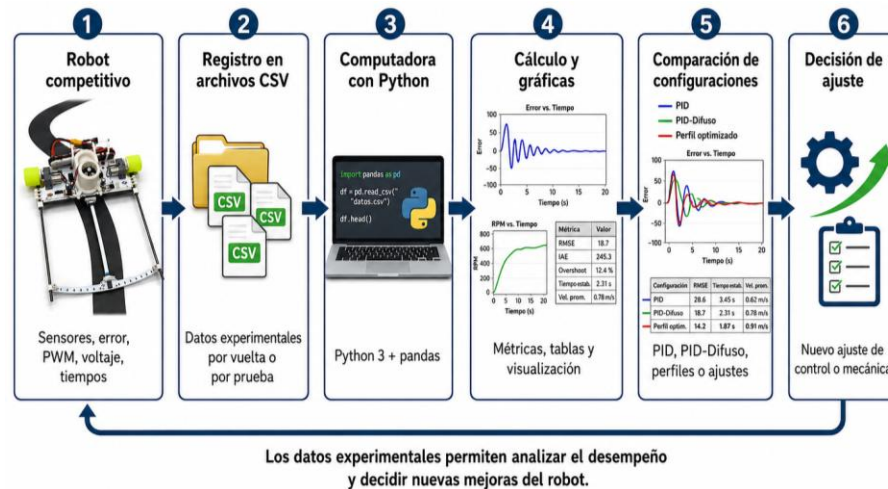


Figura 19.2: Flujo de trabajo para análisis experimental de robots competitivos.

19.4. Aplicación STEM

Este proyecto integra las áreas STEM de la siguiente forma:

- **Ciencia:** observación, medición, hipótesis, repetibilidad y validación experimental.
- **Tecnología:** Python, pandas, matplotlib, CSV, microSD y telemetría.
- **Ingeniería:** toma de decisiones para mejorar desempeño de robots.
- **Matemáticas:** promedio, máximo, mínimo, desviación, comparación porcentual e interpretación gráfica.

19.5. Objetivos del proyecto

19.5.1. Objetivo general

Diseñar una metodología de análisis de datos experimentales con Python para evaluar y optimizar robots competitivos a partir de archivos CSV.

19.5.2. Objetivos específicos

1. Leer archivos CSV generados por robots o estaciones de cronometraje.
2. Limpiar y revisar datos experimentales.
3. Calcular métricas de desempeño.

- 4. Graficar variables relevantes.
- 5. Comparar perfiles de control.
- 6. Generar reportes simples de resultados.
- 7. Proponer decisiones de ajuste basadas en evidencia.

19.6. Materiales y herramientas

Tabla 19.1: Herramientas necesarias para análisis de datos.

Elemento	Cantidad	Función
Computadora	1	Procesamiento de archivos CSV
Python 3	1	Lenguaje de análisis
pandas	1	Lectura y manipulación de datos
matplotlib	1	Generación de gráficas
Archivos CSV	Varios	Datos experimentales
Hoja de cálculo	Opcional	Revisión manual de datos
Robot competitivo	1 o más	Fuente de datos

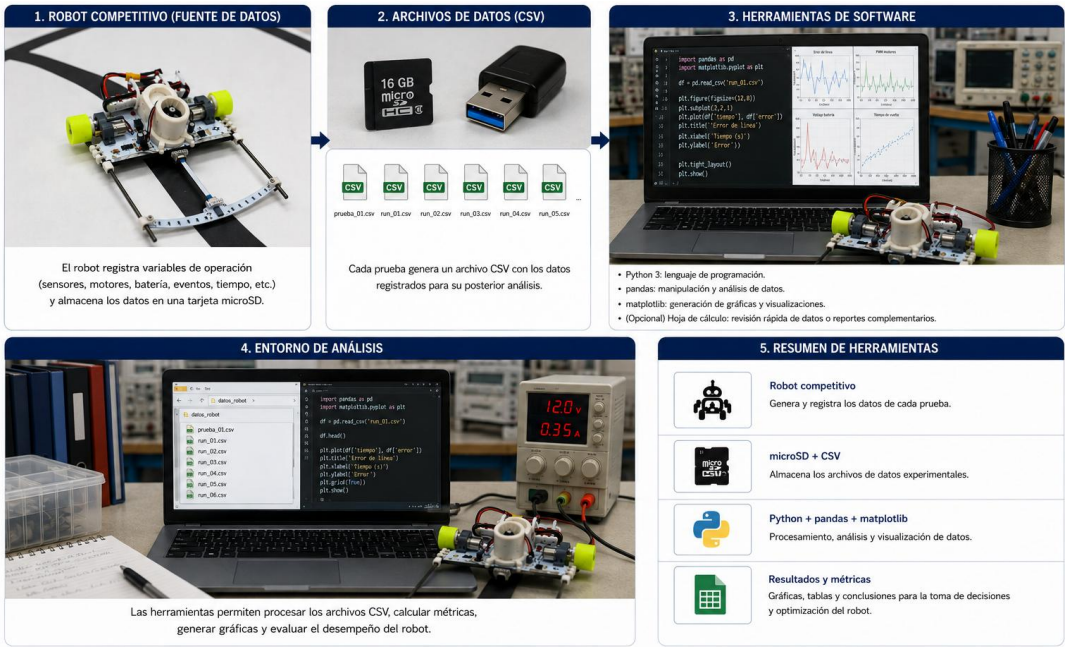


Figura 19.3: Herramientas de análisis de datos experimentales.

19.7. Estructura recomendada de archivos

Para evitar confusión, se recomienda organizar las pruebas por fecha, robot y perfil.

```
pruebas_robotica/  
Shippu/  
2026-06-25/
```

```
PID_base_LOG00.csv
PID_rapido_LOG01.csv
PID_difuso_LOG02.csv
notas_prueba.txt
Kurogane/
2026-06-25/
Ataque_agresivo_LOG00.csv
Ataque_conservador_LOG01.csv
```

El archivo de notas puede incluir:

- Voltaje inicial de batería.
- Valores de K_p , K_i , K_d .
- Velocidad base.
- Tipo de llantas.
- Condición de pista.
- Observaciones visuales.

El nombre del archivo debe decir qué se probó. Un nombre como LOG03.CSV es útil para el robot, pero no para el análisis posterior si no se documenta.

19.8. Lectura de un archivo CSV

El siguiente ejemplo lee un archivo CSV generado por el data logger del Capítulo 18 [2].

```
import pandas as pd

archivo = "PID_difuso_LOG02.csv"

datos = pd.read_csv(archivo)

print(datos.head())
print(datos.columns)
print(datos.describe())
```

Si el archivo contiene columnas:

```
muestra, tiempo_ms, voltaje, corriente, estado, error, pwm_izq, pwm_der, eventos
```

entonces se pueden analizar directamente las variables principales.

19.9. Revisión inicial de datos

Antes de calcular métricas, se debe verificar que el archivo sea válido:

- Que no esté vacío.
- Que tenga encabezados.
- Que el tiempo aumente correctamente.
- Que no existan valores imposibles.

- Que las columnas tengan nombres esperados.

```
import pandas as pd

datos = pd.read_csv("PID_difuso_LOG02.csv")
print("Numero de muestras:", len(datos))
print("Columnas:", list(datos.columns))

# Verificar valores nulos
print(datos.isnull().sum())

# Verificar rango de tiempo
tiempo_inicial = datos["tiempo_ms"].iloc[0]
tiempo_final = datos["tiempo_ms"].iloc[-1]

print("Tiempo inicial:", tiempo_inicial)
print("Tiempo final:", tiempo_final)
print("Duracion de prueba:", tiempo_final - tiempo_inicial, "ms")
```

19.10. Métricas de desempeño

19.10.1. Error máximo y error promedio

Para seguidores de línea, el error de línea es una variable crítica [6].

$$e_{max} = \max(|e|)$$

$$e_{prom} = \frac{1}{n} \sum_{i=1}^n |e_i|$$

```
import pandas as pd

datos = pd.read_csv("PID_difuso_LOG02.csv")

error_abs = datos["error"].abs()

error_max = error_abs.max()
error_prom = error_abs.mean()

print("Error maximo:", error_max)
print("Error promedio:", error_prom)
```

19.10.2. Saturación de PWM

La saturación ocurre cuando el PWM llega al límite máximo. Esto puede indicar que el controlador está exigiendo más corrección de la que el robot puede entregar.

$$sat = \frac{N_{saturadas}}{N_{total}} \cdot 100$$

```
PWM_LIMITE = 240

sat_izq = (datos["pwm_izq"].abs() >= PWM_LIMITE).mean() * 100
sat_der = (datos["pwm_der"].abs() >= PWM_LIMITE).mean() * 100

print("Saturacion izquierda:", sat_izq, " %")
print("Saturacion derecha:", sat_der, " %")
```

19.10.3. Caída de voltaje

Una caída fuerte de voltaje puede explicar pérdida de potencia o reinicios.

$$\Delta V = V_{max} - V_{min}$$

```
vmax = datos["voltaje"].max()
vmin = datos["voltaje"].min()
caida = vmax - vmin

print("Voltaje maximo:", vmax)
print("Voltaje minimo:", vmin)
print("Caída de voltaje:", caida)
```

19.10.4. Eventos de falla

Si el archivo registra eventos, se puede calcular cuántas veces ocurrieron:

```
eventos_final = datos["eventos"].iloc[-1]
print("Eventos registrados:", eventos_final)
```

19.11. Gráficas básicas

Las gráficas permiten entender el comportamiento del robot [3].

19.11.1. Error de línea en el tiempo

```
import pandas as pd
import matplotlib.pyplot as plt

datos = pd.read_csv("PID_difuso_LOG02.csv")

t = datos["tiempo_ms"] / 1000.0

plt.figure()
plt.plot(t, datos["error"])
plt.xlabel("Tiempo (s)")
plt.ylabel("Error de línea")
plt.title("Error de línea en el tiempo")
plt.grid(True)
plt.show()
```

19.11.2. PWM de motores

```
plt.figure()
plt.plot(t, datos["pwm_izq"], label="PWM izquierdo")
plt.plot(t, datos["pwm_der"], label="PWM derecho")
plt.xlabel("Tiempo (s)")
plt.ylabel("PWM")
plt.title("PWM de motores")
plt.legend()
plt.grid(True)
plt.show()
```

19.11.3. Voltaje de batería

```
plt.figure()
plt.plot(t, datos["voltaje"])
plt.xlabel("Tiempo (s)")
plt.ylabel("Voltaje (V)")
plt.title("Voltaje de batería durante la prueba")
plt.grid(True)
plt.show()
```

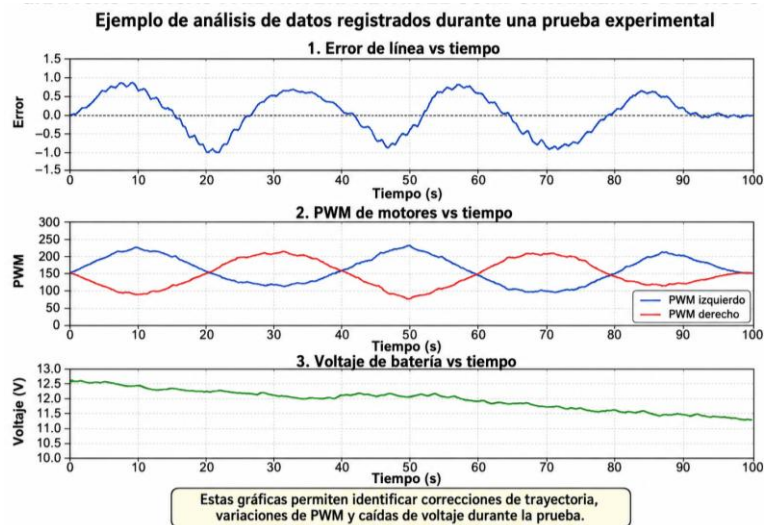


Figura 19.4: Gráficas básicas para interpretar el comportamiento del robot.

19.12. Comparación de dos perfiles

Para comparar dos perfiles, se cargan dos archivos CSV y se calculan métricas equivalentes.

```
import pandas as pd

def analizar_archivo(nombre):
    datos = pd.read_csv(nombre)
```



```

resultado = {
    "archivo": nombre,
    "muestras": len(datos),
    "error_max": datos["error"].abs().max(),
    "error_prom": datos["error"].abs().mean(),
    "voltaje_min": datos["voltaje"].min(),
    "corriente_max": datos["corriente"].max(),
    "eventos": datos["eventos"].iloc[-1]
}

return resultado

perfil_a = analizar_archivo("PID_base_LOG00.csv")
perfil_b = analizar_archivo("PID_difuso_LOG02.csv")

print(perfil_a)
print(perfil_b)

```

19.13. Comparación de varios perfiles

Cuando existen varios archivos, se puede automatizar el análisis.

```

import pandas as pd
from pathlib import Path

def analizar_archivo(ruta):
    datos = pd.read_csv(ruta)

    return {
        "archivo": ruta.name,
        "muestras": len(datos),
        "duracion_s": (datos["tiempo_ms"].iloc[-1] - datos["tiempo_ms"].iloc[0]) /
            1000,
        "error_max": datos["error"].abs().max(),
        "error_prom": datos["error"].abs().mean(),
        "voltaje_min": datos["voltaje"].min(),
        "corriente_max": datos["corriente"].max(),
        "eventos": datos["eventos"].iloc[-1]
    }

carpeta = Path("pruebas")
resultados = []

for archivo in carpeta.glob("*.csv"):
    resultados.append(analizar_archivo(archivo))

tabla = pd.DataFrame(resultados)
tabla = tabla.sort_values("error_prom")

```

```
print(tabla)
tabla.to_csv("resumen_perfiles.csv", index=False)
```

19.14. Tabla de decisión

El análisis debe terminar en una decisión. Una tabla de decisión permite comparar ventajas y riesgos.

Tabla 19.2: Ejemplo de tabla de decisión para perfiles de seguidor de línea.

Perfil	Tiempo	Error prom.	Eventos	Decisión
PID base	Medio	Bajo	0	Seguro
PID rápido	Bajo	Alto	2	Riesgoso
PID-Difuso	Bajo	Medio	0	Recomendado
PID técnico	Medio	Muy bajo	0	Para pistas cerradas

El perfil recomendado no siempre es el que tiene menor tiempo. Debe equilibrar rapidez, estabilidad, consumo y probabilidad de completar la pista.

19.15. Análisis específico para seguidores de línea

Para seguidores de línea se recomienda analizar [7]:

- Error absoluto promedio.
- Error máximo.
- Saturación de PWM.
- Tiempo con error alto.
- Recuperaciones de línea.
- Voltaje mínimo durante la vuelta.

```
UMBRAL_ERROR_ALTO = 4500

porcentaje_error_alto = (datos["error"].abs() > UMBRAL_ERROR_ALTO).mean() * 100

print("Tiempo con error alto:", porcentaje_error_alto, " %")
```

Si un perfil tiene poco tiempo total pero mucho porcentaje de error alto, puede estar al límite de salirse.

19.16. Análisis específico para Minisumo

Para Minisumo se recomienda analizar:

- Tiempo en estado de búsqueda.
- Tiempo en ataque.

- Número de evasiones de borde.
- Corriente máxima durante empuje.
- Caída de voltaje al atacar.
- Activación de sensores de oponente.

```
conteo_estados = datos["estado"].value_counts()
print(conteo_estados)
```

Si el robot pasa demasiado tiempo en búsqueda, puede necesitar mejor estrategia de giro o sensores de mayor cobertura.

19.17. Generación de un reporte simple

El siguiente código genera un resumen en texto para documentar una prueba.

```
import pandas as pd

archivo = "PID_difuso_LOG02.csv"
datos = pd.read_csv(archivo)

reporte = []

reporte.append("REPORTE DE PRUEBA")
reporte.append("=====")
reporte.append(f"Archivo: {archivo}")
reporte.append(f"Muestras: {len(datos)}")
reporte.append(f"Error máximo: {datos['error'].abs().max()}")
reporte.append(f"Error promedio: {datos['error'].abs().mean():.2f}")
reporte.append(f"Voltaje mínimo: {datos['voltaje'].min():.2f} V")
reporte.append(f"Corriente máxima: {datos['corriente'].max():.2f} A")
reporte.append(f"Eventos: {datos['eventos'].iloc[-1]}")

with open("reporte_prueba.txt", "w", encoding="utf-8") as f:
    for linea in reporte:
        f.write(linea + "\n")

print("Reporte generado")
```

19.18. Metodología de optimización experimental

Una metodología recomendada es [5]:

1. Definir un objetivo: reducir tiempo, reducir oscilación o mejorar estabilidad.
2. Establecer una configuración base.
3. Cambiar un solo parámetro.
4. Ejecutar varias pruebas.
5. Registrar datos.

6. Calcular métricas.
7. Comparar con la base.
8. Aceptar, rechazar o ajustar el cambio.



Figura 19.5: Ciclo de optimización experimental basado en datos.

19.19. Ejemplo de interpretación

Supóngase que se comparan dos perfiles:

- Perfil A: velocidad 120, $K_p = 0.045$, $K_d = 0.75$.
- Perfil B: velocidad 145, $K_p = 0.045$, $K_p = 0.090$.

El Perfil B reduce el tiempo de vuelta en 4 %, pero duplica el porcentaje de error alto y presenta dos eventos de pérdida de línea. En entrenamiento puede parecer atractivo, pero en competencia sería riesgoso. Una decisión razonable sería probar una velocidad intermedia o aumentar la reducción de velocidad en curvas.

Un mejor tiempo acompañado de mayor inestabilidad no siempre significa mejora real. La competencia premia completar la pista de forma rápida, pero también confiable.

19.20. Errores frecuentes

Durante el análisis pueden presentarse los siguientes errores:

- Comparar pruebas con baterías en diferente estado de carga.
- Mezclar archivos sin documentar qué perfil corresponde a cada uno.
- Analizar una sola vuelta y sacar conclusiones definitivas.
- Cambiar varios parámetros al mismo tiempo.
- Confundir correlación con causa.

- Ignorar eventos de falla porque el mejor tiempo fue bueno.
- No revisar si el archivo CSV tiene datos incompletos.
- Usar gráficas sin unidades o sin títulos claros.

19.21. Proyecto de extensión

Como extensión, se propone crear un script que lea automáticamente una carpeta completa de pruebas, genere gráficas, cree una tabla resumen y exporte un reporte en PDF o HTML.

Otra extensión consiste en integrar el análisis con un dashboard. El robot podría enviar datos en vivo, guardarlos localmente y luego procesarlos automáticamente al terminar la prueba.



Figura 19.6: Extensión con generación automática de reportes de prueba.

Actividades de aprendizaje

1. Abra un archivo CSV generado por el data logger y muestre sus primeras filas.
2. Calcule error máximo, error promedio y voltaje mínimo.
3. Grafique error de línea y PWM de motores.
4. Compare dos perfiles de control usando Python.
5. Genere una tabla resumen de varios archivos CSV.
6. Redacte una decisión de ajuste basada en los resultados obtenidos.

Preguntas de reflexión

1. ¿Por qué una gráfica puede revelar problemas que no se observan a simple vista?
2. ¿Qué métrica usaría para evaluar estabilidad en un seguidor de línea?

3. ¿Qué métrica usaría para evaluar agresividad en un Minisumo?
4. ¿Por qué no conviene cambiar varios parámetros al mismo tiempo?
5. ¿Cómo decidiría entre un perfil rápido pero riesgoso y uno estable pero más lento?
6. ¿Cómo aplicaría este análisis al ajuste de un seguidor de línea en una pista con curvas pequeñas?

Referencias

- [1] Python Software Foundation, “Python 3 Documentation”. [En línea]. Disponible en: <https://docs.python.org/3/>. [Accedido: 25-jun-2026].
- [2] pandas Development Team, “pandas Documentation”. [En línea]. Disponible en: <https://pandas.pydata.org/docs/>. [Accedido: 25-jun-2026].
- [3] Matplotlib Development Team, “Matplotlib Documentation”. [En línea]. Disponible en: <https://matplotlib.org/stable/contents.html>. [Accedido: 25-jun-2026].
- [4] NumPy Developers, “NumPy Documentation”. [En línea]. Disponible en: <https://numpy.org/doc/>. [Accedido: 25-jun-2026].
- [5] D. C. Montgomery, *Design and Analysis of Experiments*, 10th ed. Hoboken, NJ, USA: Wiley, 2019.
- [6] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ, USA: Princeton University Press, 2008.
- [7] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.
- [8] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2017.

Capítulo 20. Aprendizaje Automático Básico para Robots Competitivos

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Comprender el papel del aprendizaje automático en robots competitivos y sistemas embebidos.
- Diferenciar entre reglas manuales, control clásico, control difuso y modelos entrenados con datos.
- Construir un conjunto de datos a partir de archivos CSV registrados en pruebas reales.
- Etiquetar estados del robot como recta, curva, pérdida de línea, ataque o evasión.
- Entrenar un clasificador simple en Python usando variables experimentales.
- Evaluar el desempeño de un modelo mediante matriz de confusión y exactitud.
- Exportar reglas simples hacia Arduino para clasificación en tiempo real.
- Reconocer límites prácticos del aprendizaje automático en microcontroladores de bajo recurso.

20.1. Introducción

En los capítulos anteriores se desarrollaron herramientas para registrar datos, analizarlos y optimizar parámetros de control. Esos datos permiten comprender cómo se comporta un robot durante una prueba. El siguiente paso natural es utilizar los datos para que el robot aprenda a reconocer situaciones.

El aprendizaje automático, conocido también como machine learning, permite construir modelos a partir de ejemplos. En lugar de escribir manualmente todas las reglas, se recopilan datos de sensores y se asocia cada muestra con una etiqueta. Por ejemplo, en un seguidor de línea se pueden etiquetar muestras como recta, curva suave, curva cerrada o pérdida de línea. En un Minisumo se pueden etiquetar estados como *búsqueda*, *oponente detectado*, *ataque*, *empuje* o *borde*.

Este capítulo introduce el aprendizaje automático de forma práctica y accesible. No se busca implementar redes neuronales complejas ni modelos pesados. El objetivo es mostrar cómo usar datos registrados por el robot para entrenar modelos simples, interpretables y exportables a Arduino.

Se trabajará principalmente con clasificadores sencillos, como árboles de decisión y reglas derivadas. Estos modelos son útiles porque pueden convertirse en condiciones if comprensibles.

De esta manera, el aprendizaje automático se convierte en una herramienta de apoyo para diseñar mejores estrategias.

El capítulo está orientado a robots competitivos reales: seguidores de línea, Minisumo, robots exploradores, bancos de motores y sistemas de telemetría. También sirve como introducción a proyectos más avanzados con ESP32, Edge AI o TinyML.

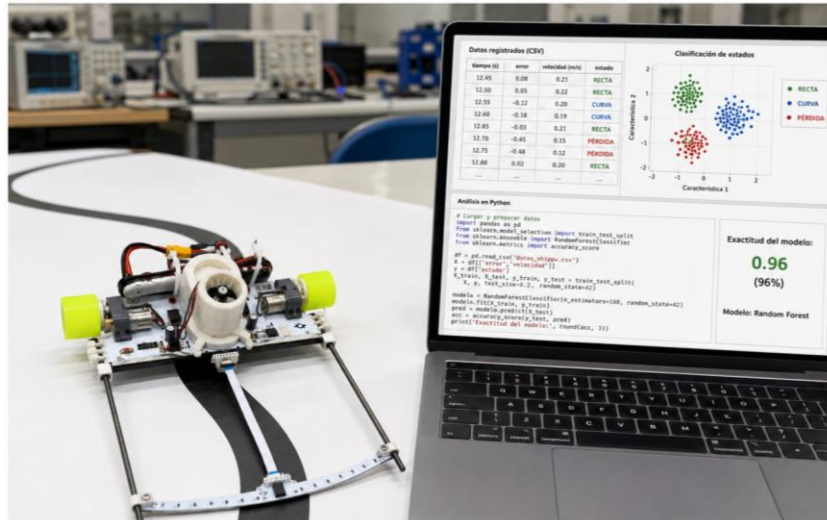


Figura 20.1: Aprendizaje automático aplicado al análisis de robots competitivos.

20.2. Planteamiento del problema

Un equipo de robótica puede observar el comportamiento del robot y describirlo manualmente:

- En recta el error es pequeño.
- En curva el error aumenta.
- Cuando se pierde la línea, la suma de sensores baja.
- En Minisumo, un empuje fuerte aumenta la corriente.
- Una evasión de borde activa sensores reflectivos.

Estas observaciones pueden convertirse en reglas. Sin embargo, cuando aumentan las variables, las reglas manuales se vuelven difíciles de ajustar. Por ejemplo, para clasificar una curva puede ser necesario considerar error, derivada, PWM, velocidad, sensores activos y estado anterior.

El problema del capítulo es construir un método para que el robot, a partir de datos, pueda reconocer estados relevantes de su comportamiento.

El aprendizaje automático no reemplaza la comprensión del robot. Usa datos para descubrir o ajustar reglas que luego deben ser revisadas por el equipo.

20.3. Descripción general del proyecto

El flujo de trabajo propuesto es:

1. Registrar datos del robot en archivo CSV.
2. Añadir etiquetas a las muestras.
3. Entrenar un clasificador en Python.
4. Evaluar el modelo.
5. Extraer reglas simples.
6. Implementar esas reglas en Arduino.
7. Probar el clasificador en el robot real.

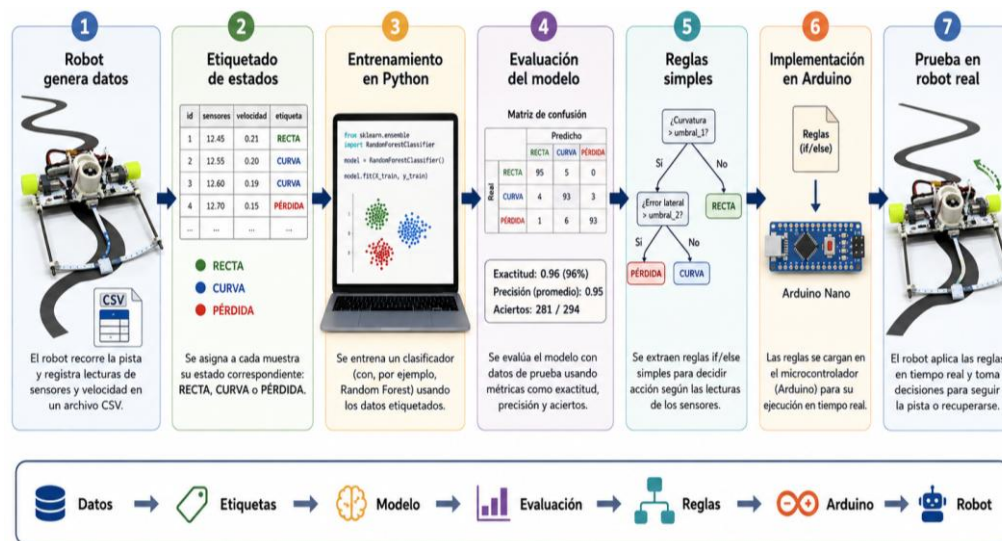


Figura 20.2: Flujo de trabajo para aprendizaje automático básico en robots competitivos.

20.4. Aplicación STEM

Este proyecto integra las áreas STEM de la siguiente manera:

- **Ciencia:** clasificación, patrones, validación experimental y análisis de errores.
- **Tecnología:** Python, CSV, sensores, microcontroladores y modelos simples.
- **Ingeniería:** diseño de estrategias inteligentes basadas en datos reales.
- **Matemáticas:** vectores de características, métricas, matrices de confusión y reglas de decisión.

20.5. Objetivos del proyecto

20.5.1. Objetivo general

Diseñar e implementar una metodología básica de aprendizaje automático para clasificar estados de robots competitivos a partir de datos registrados en pruebas reales.

20.5.2. Objetivos específicos

- 1. Construir un conjunto de datos con variables internas del robot.
- 2. Definir etiquetas útiles para seguidores de línea y Minisumo.
- 3. Entrenar un clasificador simple en Python.
- 4. Evaluar el desempeño del modelo.
- 5. Interpretar las reglas generadas.
- 6. Implementar una versión simplificada del clasificador en Arduino.
- 7. Analizar limitaciones de memoria, tiempo de cálculo y generalización.

20.6. Materiales y herramientas

Tabla 20.1: Herramientas necesarias para aprendizaje automático básico.

Elemento	Cantidad	Función
Robot competitivo	1	Fuente de datos experimentales
Data logger microSD o telemetría	1	Registro de datos
Computadora	1	Entrenamiento y análisis
Python 3	1	Programación del modelo
pandas	1	Manejo de datos CSV
scikit-learn	1	Entrenamiento de clasificadores
matplotlib	1	Visualización de resultados
Arduino IDE	1	Implementación embebida

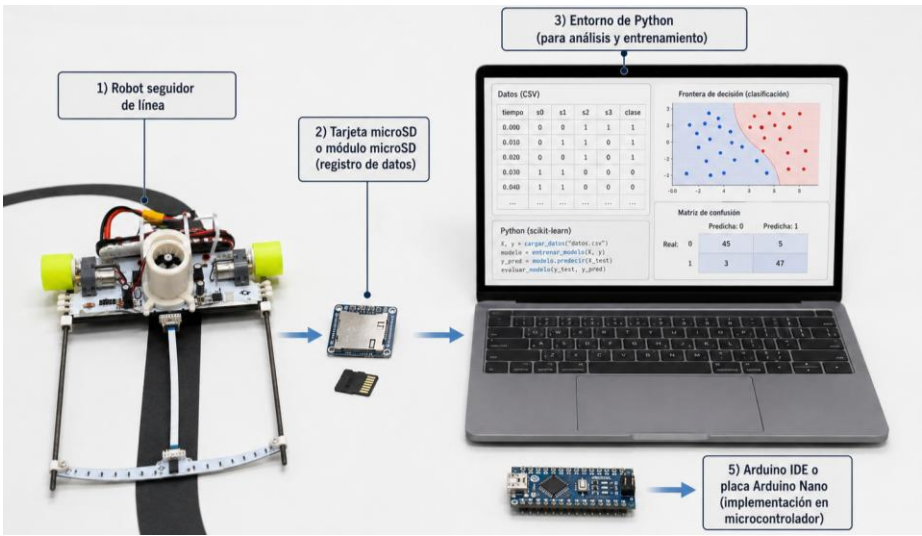


Figura 20.3: Herramientas para aprendizaje automático aplicado a robots.

20.7. Fundamento teórico

20.7.1. Características

Las características son las variables que describen una muestra. En un seguidor de línea pueden ser:

- Error de línea.
- Derivada del error.
- Suma de sensores.
- PWM izquierdo.
- PWM derecho.
- Voltaje de batería.
- Cada muestra puede representarse como un vector:

$$x = [\text{error}, \Delta_e, \text{sumaSensores}, \text{pwmIzq}, \text{pwmDer}, \text{voltaje}]$$

20.7.2. Etiqueta

La etiqueta es la clase que se desea predecir. Por ejemplo:

- Recta.
- Curva suave.
- Curva cerrada.
- Pérdida de línea.

El aprendizaje supervisado utiliza pares:

$$(x_i, y_i)$$

donde x_i es el vector de características y y_i es la etiqueta.

20.7.3. Clasificador

Un clasificador aprende una relación entre características y etiquetas. Después del entrenamiento, puede recibir una muestra nueva y predecir su clase [1, 2].

$$\hat{y} = f(x)$$

20.7.4. Árbol de decisión

Un árbol de decisión clasifica mediante preguntas sucesivas. Por ejemplo:

Si $|\text{error}| < 800$, entonces recta

Si $|\text{error}| > 4500$ y sumaSensores baja, entonces pérdida de línea

Los árboles son útiles para robótica embebida porque pueden convertirse en reglas if.

20.7.5. Matriz de confusión

La matriz de confusión compara etiquetas reales y etiquetas predichas. Permite identificar qué clases se confunden.

$$\text{Exactitud} = \frac{\text{predicciones correctas}}{\text{total de predicciones}}$$

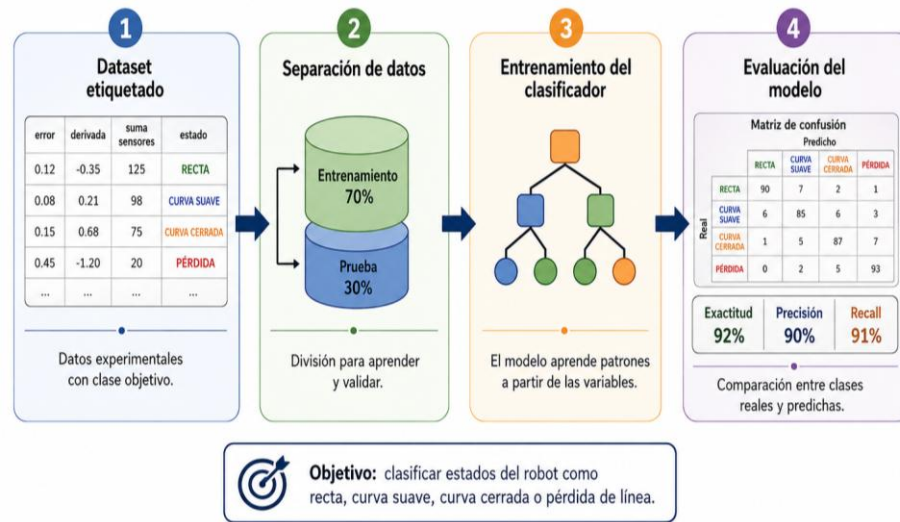


Figura 20.4: Proceso básico de entrenamiento y evaluación de un clasificador.

20.8. Construcción del conjunto de datos

El archivo CSV debe contener variables útiles. Para un seguidor de línea se propone:

```
tiempo_ms,error,derivada,suma_sensores,pwm_izq,pwm_der,voltaje,estado
```

La columna estado contiene la etiqueta. Puede añadirse manualmente después de revisar la prueba o generarse a partir de marcas de pista.

Tabla 20.2: Etiquetas sugeridas para seguidor de línea.

Etiqueta	Descripción	Uso estratégico
RECTA	Error bajo y estable	Aumentar velocidad
CURVA_SUAVE	Error medio	Corrección normal
CURVA_CERRADA	Error alto	Reducir velocidad
PERDIDA	Poca lectura de línea	Recuperar línea
OSCILACION	Error cambia de signo rápido	Suavizar control

Para Minisumo se propone:

```
tiempo_ms,sensor_izq,sensor_cen,sensor_der,borde_izq,borde_der,corriente,voltaje,pwm_izq,pwm_d
```

Tabla 20.3: Etiquetas sugeridas para Minisumo.

Etiqueta	Descripción	Uso estratégico
BUSQUEDA	No hay oponente detectado	Girar o explorar
DETECCION	Oponente visible	Alinear ataque
ATAQUE	Oponente centrado	Avanzar rápido
EMPUJE	Corriente alta y avance sostenido	Mantener potencia
BORDE	Sensor de borde activo	Evasión inmediata

20.9. Etiquetado de datos

El etiquetado puede realizarse de tres formas:

1. **Manual:** revisar video y archivo CSV para asignar etiquetas.
2. **Por segmentos:** marcar intervalos de tiempo conocidos, por ejemplo: curva entre 5 s y 7 s.
3. **Por reglas iniciales:** usar umbrales simples para crear etiquetas preliminares.

El etiquetado manual es más lento, pero puede ser más confiable. El etiquetado por reglas es más rápido, pero puede introducir errores.

El modelo aprende de las etiquetas. Si las etiquetas están mal, el modelo también aprenderá mal. La calidad del conjunto de datos es más importante que la complejidad del algoritmo.

20.10. Etiquetado inicial con Python

El siguiente ejemplo crea etiquetas preliminares para un seguidor de línea usando reglas simples [5, 6].

```
import pandas as pd

datos = pd.read_csv("LOG_SEGUIDOR.csv")

datos["error_abs"] = datos["error"].abs()
datos["derivada_abs"] = datos["derivada"].abs()

def etiquetar(fila):
    if fila["suma_sensores"] < 200:
        return "PERDIDA"

    if fila["error_abs"] < 800 and fila["derivada_abs"] < 500:
        return "RECTA"

    if fila["error_abs"] < 3000:
        return "CURVA_SUAVE"
```

```

return "CURVA_CERRADA"

datos["estado"] = datos.apply(etiquetar, axis=1)

datos.to_csv("dataset_seguidor_etiquetado.csv", index=False)

print(datos["estado"].value_counts())

```

20.11. Entrenamiento de un árbol de decisión

El siguiente código entrena un árbol de decisión para clasificar estados [3, 4].

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix

datos = pd.read_csv("dataset_seguidor_etiquetado.csv")

caracteristicas = ["error", "derivada", "suma_sensores", "pwm_izq", "pwm_der",
                  "voltaje"]
X = datos[caracteristicas]
y = datos["estado"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=42, stratify=y
)

modelo = DecisionTreeClassifier(
    max_depth=4,
    random_state=42
)

modelo.fit(X_train, y_train)

pred = modelo.predict(X_test)

print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

```

20.12. Visualización del árbol

Un árbol de decisión puede visualizarse para entender qué reglas aprendió.

```

from sklearn.tree import export_text

reglas = export_text(modelo, feature_names=caracteristicas)

```

```
print(reglas)

with open("reglas_arbol.txt", "w", encoding="utf-8") as f:
    f.write(reglas)
```

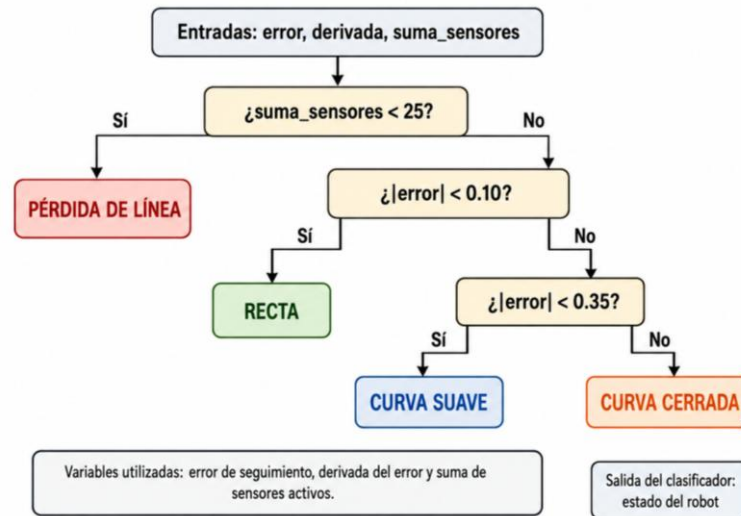


Figura 20.5: Árbol de decisión entrenado para clasificación de estados.

20.13. Interpretación de reglas

Un árbol de decisión puede producir reglas como:

```
if suma_sensores <= 180:
    estado = PERDIDA
else:
    if abs(error) <= 900:
        estado = RECTA
    else:
        if abs(error) <= 3200:
            estado = CURVA_SUAVE
        else:
            estado = CURVA_CERRADA
```

Estas reglas son fáciles de implementar en Arduino. La ventaja no está en que sean imposibles de escribir manualmente, sino en que fueron derivadas a partir de datos reales y pueden ajustarse con nuevas pruebas.

20.14. Implementación en Arduino

Una vez interpretadas las reglas, se puede programar un clasificador embebido [7].

```
enum EstadoLinea
```

```

{
    RECTA,
    CURVA_SUAVE,
    CURVA_CERRADA,
    PERDIDA,
    OSCILACION
};

EstadoLinea clasificarEstado(long error, long derivada, int sumaSensores)
{
    long errorAbs = abs(error);
    long derivadaAbs = abs(derivada);

    if (sumaSensores < 200)
    {
        return PERDIDA;
    }

    if (errorAbs < 800 && derivadaAbs < 500)
    {
        return RECTA;
    }

    if (errorAbs < 3000)
    {
        return CURVA_SUAVE;
    }

    if (derivadaAbs > 4000)
    {
        return OSCILACION;
    }

    return CURVA_CERRADA;
}

void aplicarEstrategia(EstadoLinea estado)
{
    switch (estado)
    {
        case RECTA:
            // aumentar velocidad

            velocidadBase = 150;
            factorCorreccion = 0.85;
            break;
    }
}

```



```

case CURVA_SUAVE:
    velocidadBase = 125;
    factorCorreccion = 1.00;
    break;

case CURVA_CERRADA:
    velocidadBase = 90;
    factorCorreccion = 1.30;
    break;

case PERDIDA:
    velocidadBase = 70;
    factorCorreccion = 1.50;
    recuperarLinea();
    break;

case OSCILACION:
    velocidadBase = 100;
    factorCorreccion = 0.80;
    break;
}
}

```

En microcontroladores pequeños, como Arduino Nano, conviene usar modelos simples. Un árbol de decisión poco profundo puede convertirse en reglas if rápidas y comprensibles.

20.15. Uso del clasificador en un seguidor de línea

El clasificador puede integrarse en el ciclo de control [8]:

1. Leer sensores.
2. Calcular error y derivada.
3. Calcular suma de sensores.
4. Clasificar estado.
5. Seleccionar velocidad y factor de corrección.
6. Ejecutar PID o PID-Difuso.

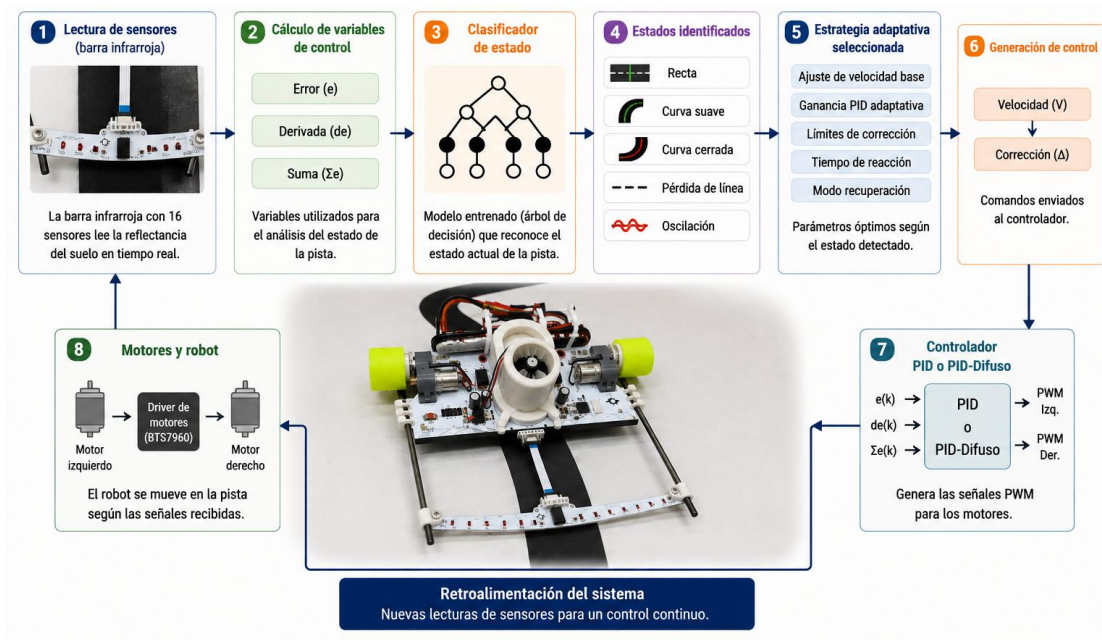


Figura 20.6: Integración del clasificador con el controlador del seguidor de línea.

20.16. Uso del clasificador en Minisumo

En Minisumo, un clasificador puede distinguir situaciones de combate:

- Búsqueda.
- Detección lateral.
- Ataque centrado.
- Empuje.
- Riesgo de borde.
- Variables útiles:
- Sensores de oponente.
- Sensores de borde.
- Corriente.
- PWM.
- Cambio de velocidad.
- Tiempo en estado actual.

```
enum EstadoCombate
{
    BUSQUEDA,
    DETECCION,
    ATAQUE,
    EMPUJE,
    BORDE
}
```

```
};

EstadoCombate clasificarCombate(int izq, int cen, int der,
bool bordeIzq, bool bordeDer,
float corriente)
{
    if (bordeIzq || bordeDer)
    {
        return BORDE;
    }

    if (corriente > 8.0 && cen > 700)
    {
        return EMPUJE;
    }

    if (cen > 600)
    {
        return ATAQUE;
    }

    if (izq > 500 || der > 500)
    {
        return DETECCION;
    }

    return BUSQUEDA;
}
```

20.17. Evaluación del modelo

La exactitud global puede ser útil, pero no suficiente. En robótica competitiva, algunas clases son más importantes que otras. Por ejemplo, clasificar mal una recta como curva puede hacer el robot más lento, pero clasificar mal una pérdida de línea como recta puede causar salida de pista.

Por ello se debe revisar:

- Exactitud por clase.
- Matriz de confusión.
- Errores críticos.
- Comportamiento en pruebas reales.

```
import pandas as pd
from sklearn.metrics import confusion_matrix

etiquetas = ["RECTA", "CURVA_SUAVE", "CURVA_CERRADA", "PERDIDA"]
```

```
matriz = confusion_matrix(y_test, pred, labels=etiquetas)

matriz_df = pd.DataFrame(
    matriz,
    index=["Real_" + e for e in etiquetas],
    columns=["Pred_" + e for e in etiquetas]
)

print(matriz_df)
```

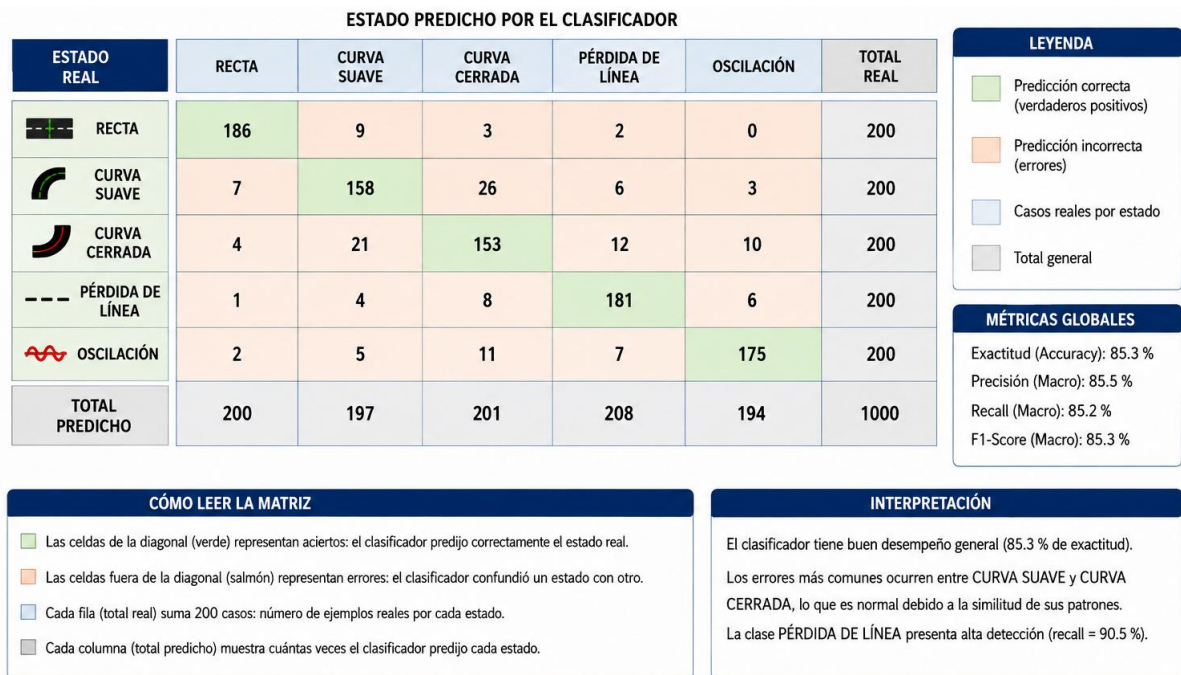


Figura 20.7: Matriz de confusión para evaluar el clasificador.

20.18. Validación en el robot real

Después de entrenar y exportar reglas, se debe probar el robot en condiciones reales:

1. Ejecutar el robot con el clasificador activo.
2. Registrar el estado predicho en la microSD.
3. Comparar estados predichos con video de la prueba.
4. Corregir reglas si aparecen errores críticos.
5. Repetir entrenamiento con datos nuevos.

Tabla 20.4: Registro sugerido de validación del clasificador.

Situación	Estado esperado	Estado predicho	Observación
Recta larga	RECTA		
Curva suave	CURVA_SUAVE		
Curva cerrada	CURVA_CERRADA		
Pérdida parcial	PERDIDA		
Oscilación	OSCILACION		

20.19. Buenas prácticas

- Empezar con pocas clases.
- Usar variables que realmente estén disponibles en el robot.
- Evitar modelos demasiado complejos.
- Separar datos de entrenamiento y prueba.
- Validar con una pista diferente.
- Registrar nuevas pruebas después de aplicar el modelo.
- Revisar errores críticos, no solo exactitud promedio.
- Mantener una versión manual de respaldo.

20.20. Limitaciones

El aprendizaje automático puede fallar si:

- Los datos de entrenamiento no representan la pista real.
- Las etiquetas están mal asignadas.
- Cambia la iluminación.
- Cambia la altura de sensores.
- Cambia la batería o velocidad.
- El modelo es demasiado complejo para el microcontrolador.
- Se usa el modelo sin validación real.

Un modelo entrenado no debe considerarse correcto solo porque funciona en los datos de entrenamiento. Debe probarse en condiciones nuevas y reales.

20.21. Proyecto de extensión

Como extensión, se propone construir un sistema de aprendizaje incremental: después de cada prueba, el robot guarda datos y el equipo actualiza el modelo con nuevas muestras. Así, el clasificador puede mejorar con diferentes pistas, iluminaciones y perfiles de velocidad.

Otra extensión es utilizar ESP32 para ejecutar modelos más complejos o integrar herramientas de TinyML. En ese caso, podrían clasificarse vibraciones, impactos, sonidos o patrones de movimiento.

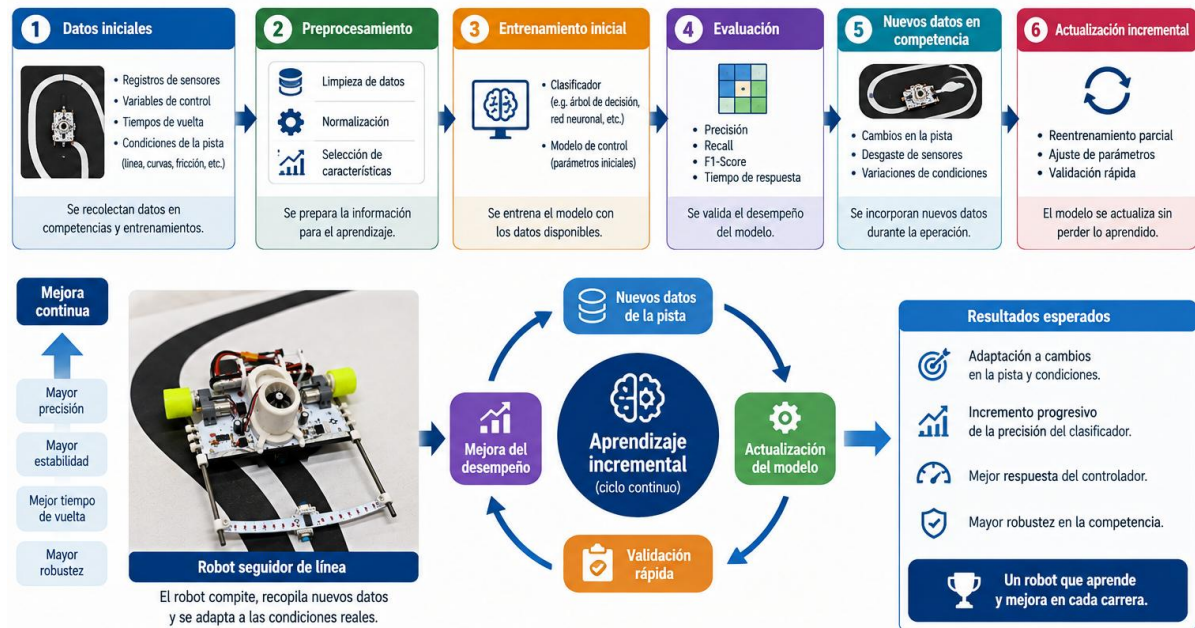


Figura 20.8: Extensión hacia aprendizaje incremental en robots competitivos.

Actividades de aprendizaje

1. Genere un archivo CSV de un seguidor de línea con error, derivada y suma de sensores.
2. Etiquete manualmente al menos cuatro estados del robot.
3. Entrene un árbol de decisión con Python.
4. Exporte las reglas del árbol y analícelas.
5. Implemente una versión simplificada del clasificador en Arduino.
6. Compare el comportamiento del robot con y sin clasificador de estados.

Preguntas de reflexión

1. ¿Qué diferencia existe entre una regla manual y una regla aprendida a partir de datos?
2. ¿Por qué es importante etiquetar correctamente las muestras?
3. ¿Qué clase sería más crítica en un seguidor de línea: recta o pérdida de línea?
4. ¿Por qué un árbol de decisión es conveniente para Arduino?
5. ¿Qué variables usaría para clasificar empuje en Minisumo?
6. ¿Cómo evitaría que el modelo funcione solo en una pista específica?

Referencias

- [1] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997.
- [2] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.
- [3] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 3rd ed. Sebastopol, CA, USA: O'Reilly Media, 2022.
- [4] scikit-learn Developers, “scikit-learn User Guide”. [En línea]. Disponible en: https://scikit-learn.org/stable/user_guide.html. [Accedido: 25-jun-2026].
- [5] pandas Development Team, “pandas Documentation”. [En línea]. Disponible en: <https://pandas.pydata.org/docs/>. [Accedido: 25-jun-2026].
- [6] Python Software Foundation, “Python 3 Documentation”. [En línea]. Disponible en: <https://docs.python.org/3/>. [Accedido: 25-jun-2026].
- [7] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2017.
- [8] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.

Capítulo 21. Integración Final y Preparación de un Robot Competitivo Inteligente

Resultados de aprendizaje

Al finalizar este capítulo, el lector será capaz de:

- Integrar sensores, actuadores, control, telemetría, registro de datos y diagnóstico en un robot competitivo completo.
- Diseñar una arquitectura modular de hardware y software para reducir errores durante el desarrollo.
- Elaborar un plan de validación experimental antes de una competencia.
- Crear una bitácora técnica para documentar cambios, parámetros y resultados.
- Aplicar una lista de verificación mecánica, eléctrica y de software.
- Preparar perfiles de competencia seguros, rápidos y conservadores.
- Organizar pruebas finales considerando batería, repuestos, herramientas, programación y reglamento.
- Comprender que un robot competitivo no se gana solo con código, sino con integración, disciplina experimental y confiabilidad.

21.1. Introducción

Los capítulos anteriores desarrollaron módulos específicos para robots competitivos: sensores, control PID, control difuso, odometría, visión, telemetría, registro de datos, análisis experimental y diagnóstico de fallas. Cada tema aporta una capacidad importante, pero un robot competitivo real no se construye con módulos aislados. El reto final es integrarlos de manera ordenada, confiable y mantenible [3, 4, 5].

En una competencia, el robot debe funcionar bajo presión, con poco tiempo de reparación, cambios de pista, iluminación variable, baterías limitadas y posibles fallas mecánicas. Un sistema que funciona en el laboratorio puede fallar si los cables están sueltos, si no se documentaron los parámetros, si se cargó una versión incorrecta del programa o si no se revisó la batería antes de la ronda.

Este capítulo presenta una metodología de integración final. Se propone una arquitectura completa para un robot competitivo inteligente, considerando hardware, software, control, seguridad, telemetría, diagnóstico y documentación. Además, se incluyen listas de verificación y procedimientos de prueba para preparar un robot antes de competir.

La idea central es que el rendimiento del robot depende tanto de la tecnología como del proceso. Un robot rápido, pero poco confiable, puede perder. Un robot bien documentado, probado y

mantenido tiene más posibilidades de completar su tarea, adaptarse a cambios y mejorar de forma continua.



Figura 21.1: Integración final y preparación de un robot competitivo.

21.2. Planteamiento del problema

Al llegar a la etapa final de un proyecto, suelen aparecer problemas que no están relacionados con una sola línea de código:

- El robot funciona en una prueba, pero falla en la siguiente.
- Se desconoce qué versión del programa está cargada.
- Los parámetros de competencia no están documentados.
- Una batería está descargada o desbalanceada.
- Un cable se suelta por vibración.
- La barra de sensores cambió de altura.
- El robot fue rápido en una pista, pero inestable en otra.
- No se sabe si la falla fue mecánica, eléctrica o de software.

Estos problemas muestran que el desarrollo de un robot competitivo no termina cuando el código compila. Termina cuando el sistema completo ha sido probado, documentado y preparado para operar de manera confiable.

La integración final convierte un conjunto de módulos funcionales en un robot competitivo confiable.

21.3. Descripción general del proyecto

El proyecto integrador consiste en construir una plataforma competitiva con los siguientes bloques:

- **Módulo de percepción:** sensores de línea, distancia, oponente, borde, IMU o visión.
- **Módulo de control:** PID, control difuso, máquina de estados o estrategia híbrida.
- **Módulo de actuación:** motores DC, servos, ESC, drivers y protecciones.
- **Módulo de seguridad:** supervisor de batería, corriente, temperatura, borde y señal.
- **Módulo de telemetría:** comunicación Serial, Bluetooth, WiFi o dashboard.
- **Módulo de registro:** microSD o memoria para guardar datos y eventos.
- **Módulo de diagnóstico:** detección de fallas y alertas.
- **Módulo de documentación:** bitácora, parámetros, versiones y checklist.

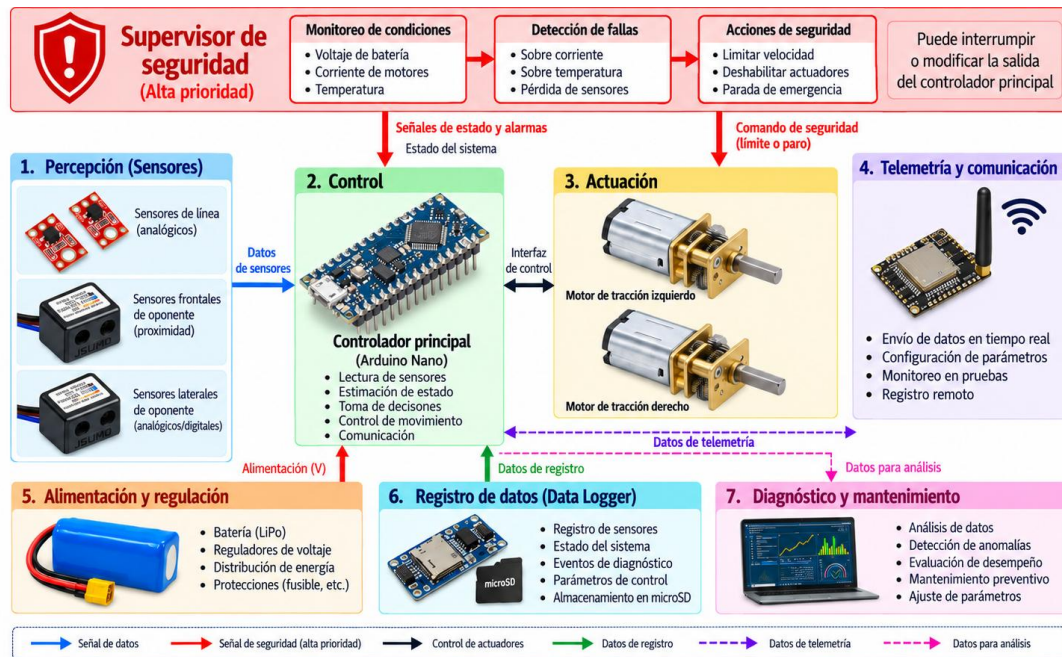


Figura 21.2: Arquitectura integral de un robot competitivo inteligente.

21.4. Aplicación STEM

Este capítulo integra las áreas STEM de forma global:

- **Ciencia:** validación, medición, pruebas repetibles y análisis de fallas.
- **Tecnología:** sensores, microcontroladores, drivers, comunicación y almacenamiento.
- **Ingeniería:** integración de sistemas, confiabilidad, mantenimiento y preparación de competencia.
- **Matemáticas:** métricas de desempeño, comparación de perfiles, tolerancias y evaluación experimental.

21.5. Objetivos del proyecto

21.5.1. Objetivo general

Diseñar una metodología de integración, validación y preparación para robots competitivos inteligentes, incorporando control, seguridad, telemetría, diagnóstico y documentación técnica.

21.5.2. Objetivos específicos

1. Definir una arquitectura modular de hardware y software.
2. Establecer prioridades de seguridad y control.
3. Crear una estructura de código organizada y mantenible.
4. Elaborar listas de verificación mecánica, eléctrica y de software.
5. Diseñar una bitácora de pruebas y parámetros.
6. Preparar perfiles de competencia.
7. Realizar pruebas finales antes de competir.
8. Documentar resultados y mantenimiento posterior.

21.6. Arquitectura modular de hardware

Una arquitectura modular facilita detectar fallas y reemplazar componentes. Se recomienda separar físicamente los sistemas [1, 8]:

- Alimentación de potencia.
- Alimentación de lógica.
- Sensores.
- Controlador principal.
- Drivers de motor.
- Comunicación.
- Registro de datos.

Tabla 21.1: Módulos físicos recomendados en un robot competitivo.

Módulo	Componentes	Criterio de diseño
Alimentación	Batería, regulador, interruptor	Capacidad, seguridad y acceso rápido
Control	Arduino, ESP32 o controlador dedicado	Ubicación protegida y cableado corto
Sensores	Línea, borde, distancia, IMU	Montaje rígido y calibrable
Potencia	Driver, ESC, MOSFETs	Disipación y rutas de corriente amplias
Telemetría	Bluetooth, WiFi, Serial	Acceso para pruebas y diagnóstico
Registro	microSD, EEPROM	Fácil extracción y protección mecánica
Registro	LED, buzzer, pantalla	Interpretación rápida en competencia

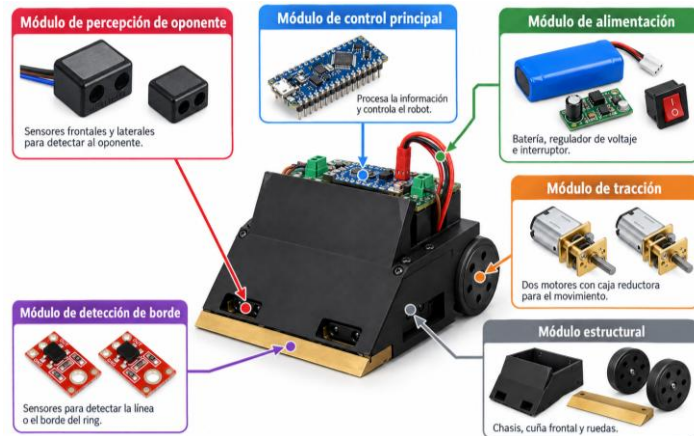


Figura 21.3: Organización modular del hardware en un robot competitivo.

21.7. Arquitectura modular de software

El software debe organizarse en funciones o archivos por responsabilidad. Aunque se programe en Arduino IDE, es posible mantener orden mediante secciones claras [2, 7].

- Lectura de sensores.
- Calibración.
- Control de motores.
- Máquina de estados.
- Seguridad. Telemetría.
- Data logger.
- Diagnóstico.
- Configuración de perfiles.

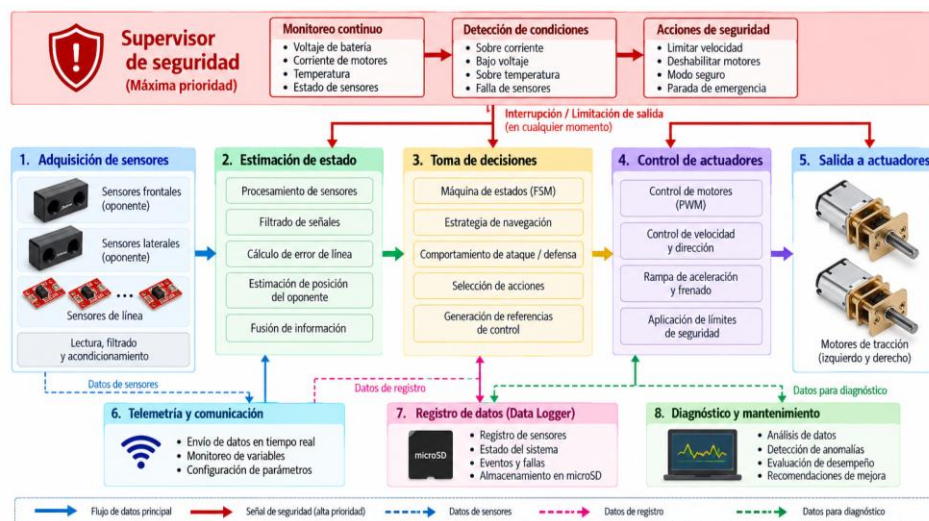


Figura 21.4: Arquitectura modular del software de control.

21.8. Prioridad de seguridad

En robots competitivos, no todas las decisiones tienen la misma prioridad. La seguridad debe estar por encima del rendimiento.

Orden recomendado:

1. Parada de emergencia o pérdida de señal.
2. Borde o condición de salida peligrosa.
3. Protección eléctrica: batería crítica, sobrecorriente o temperatura.
4. Diagnóstico de falla crítica.
5. Estrategia principal de competencia.
6. Telemetría y registro de datos.

Si una condición crítica aparece, el robot debe priorizar protección, aunque el controlador principal indique avanzar o atacar.

21.9. Estructura base de código

El siguiente esqueleto muestra una organización modular para un robot competitivo. Puede adaptarse a seguidores de línea, Minisumo o exploradores.

```
/*
Estructura modular para robot competitivo
Sensores + Control + Seguridad + Telemetria + Registro
*/

// =====
// CONFIGURACION GENERAL
// =====

#define MODO_COMPETENCIA 1
#define MODO_DEBUG 0

// Perfiles
enum PerfilRobot
{
    PERFIL_SEGURO,
    PERFIL_RAPIDO,
    PERFIL_TECNICO
};

PerfilRobot perfilActual = PERFIL_SEGURO;

// Estados generales
enum EstadoRobot
{
```

```

    INICIALIZANDO,
    LISTO,
    COMPITIENDO,
    ADVERTENCIA,
    FALLA,
    PROTECCION
};

EstadoRobot estadoRobot = INICIALIZANDO;

// Variables principales
float voltajeBateria = 0;
float corriente = 0;
float temperatura = 0;
long errorControl = 0;
long derivadaError = 0;

int pwmIzq = 0;
int pwmDer = 0;

// =====
// SETUP
// =====
void setup()
{
    iniciarComunicacion();
    iniciarPines();
    iniciarSensores();
    iniciarMotores();
    iniciarDiagnostico();
    iniciarTelemetria();
    iniciarRegistro();

    cargarPerfil(perfilActual);

    estadoRobot = LISTO;
}

// =====
// LOOP PRINCIPAL
// =====
void loop()
{
    leerSensores();
    actualizarEstimaciones();
    evaluarDiagnostico();

```

```

if (hayFallaCritica())
{
    ejecutarProteccion();
    enviarTelemetria();
    registrarDatos();
    return;
}
ejecutarEstrategia();
aplicarMotores();

enviarTelemetria();
registrarDatos();
}

// =====
// MODULOS CONCEPTUALES
// =====
void iniciarComunicacion()
{
    Serial.begin(115200);
}

void iniciarPines()
{
    // Configurar pines de sensores, motores e indicadores
}

void iniciarSensores()
{
    // Calibrar o inicializar sensores
}

void iniciarMotores()
{
    // Configurar driver y detener motores
}

void iniciarDiagnostico()
{
    // Inicializar umbrales y variables de seguridad
}

void iniciarTelemetria()
{
    // Iniciar Bluetooth, WiFi o Serial si aplica
}

```

```

void iniciarRegistro()
{
    // Iniciar microSD o EEPROM si aplica
}

void cargarPerfil(PerfilRobot perfil)
{
    switch (perfil)
    {
        case PERFIL_SEGURO:
            // Parametros conservadores
            break;

        case PERFIL_RAPIDO:
            // Parametros agresivos
            break;

        case PERFIL_TECNICO:
            // Parametros para pista compleja
            break;
    }
}

void leerSensores()
{
    // Leer sensores del robot
}

void actualizarEstimaciones()
{
    // Calcular error, estado, posicion, velocidad o diagnostico
}

void evaluarDiagnostico()
{
    // Revisar voltaje, corriente, temperatura y sensores
}

bool hayFallaCritica()
{
    // Retornar true si existe condicion peligrosa
    return false;
}

void ejecutarProteccion()
{
    detenerMotores();
}

```



```

    estadoRobot = PROTECCION;
}

void ejecutarEstrategia()
{
    // PID, difuso, maquina de estados o TinyML
}

void aplicarMotores()
{
    // Enviar pwmIzq y pwmDer al driver
}

void detenerMotores()
{
    pwmIzq = 0;
    pwmDer = 0;
    // Orden real de parada al driver
}

void enviarTelemetria()
{
    // Enviar datos si corresponde
}

void registrarDatos()
{
    // Guardar datos si corresponde
}

```

21.10. Perfiles de competencia

Un robot debe tener perfiles preparados. No siempre conviene usar el perfil más rápido.

Tabla 21.2: Perfiles sugeridos para competencia.

Perfil	Objetivo	Ventaja	Riesgo
Seguro	Terminar sin fallas	Alta confiabilidad	Menor velocidad
Rápido	Mejor tiempo o ataque agresivo	Alto rendimiento	Mayor riesgo
Técnico	Pistas complejas o curvas cerradas	Mejor control	Requiere ajuste fino
Diagnóstico	Pruebas y revisión	Facilita análisis	No usar en ronda final

21.11. Bitácora técnica

La bitácora permite saber qué se probó, qué funcionó y qué debe repetirse. Puede llevarse en papel, hoja de cálculo o archivo de texto [6].

Tabla 21.3: Formato sugerido de bitácora de pruebas.

Fecha	Robot	Perfil	Cambio realizado	Resultado
-------	-------	--------	------------------	-----------

Datos recomendados:

- Nombre del robot.
- Fecha y lugar de prueba.
- Tipo de pista o dohyo.
- Voltaje inicial de batería.
- Versión de código.
- Parámetros principales.
- Resultado observado.
- Archivo CSV asociado.
- Decisión tomada.

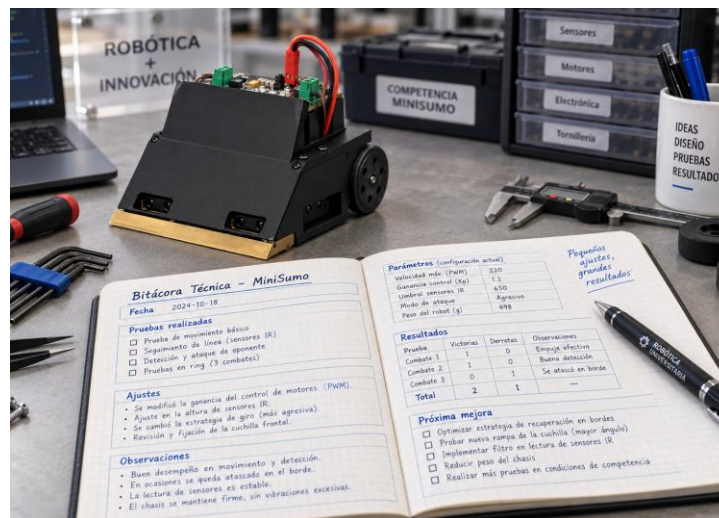


Figura 21.5: Bitácora técnica para seguimiento del desarrollo del robot.

21.12. Control de versiones

Una falla común es no saber qué código está cargado en el robot. Para evitarlo:

- Usar nombres de archivo con fecha y versión.
- Guardar una copia del código antes de cada competencia.

- Documentar cambios importantes.
- No modificar el código final sin guardar la versión estable.
- Tener una versión de respaldo.

Ejemplo de nombres:

```
Shippu_PID_Difuso_v1_2026_06_25.ino
Shippu_PID_Difuso_v2_KdMayor_2026_06_25.ino
Kurogane_MiniSumo_AtaqueSeguro_v1.ino
Kurogane_MiniSumo_AtaqueRapido_v2.ino
```

21.13. Validación mecánica

Tabla 21.4: Checklist mecánico antes de competir.

Elemento	Verificación
Tornillos	Ajustados y con fijación si aplica
Ruedas	Limpias, alineadas y sin juego excesivo
Motores	Sujetos firmemente
Chasis	Sin fisuras o piezas flojas
Sensores	Altura y orientación correctas
Batería	Firme y sin moverse
Cables	Sujetos y lejos de partes móviles
Centro de masa	Verificado según categoría

21.14. Validación eléctrica

Tabla 21.5: Checklist eléctrico antes de competir.

Elemento	Verificación
Batería	Cargada y balanceada
Voltaje	Dentro del rango permitido
Interruptor	Accesible y confiable
Regulador	Entrega voltaje estable
GND común	Verificado entre módulos
Conectores	Firmes y sin falsos contactos
Driver	Sin calentamiento anormal
Sensores	Lecturas estables
Fusibles / protecciones	Revisados si aplica

21.15. Validación de software

Tabla 21.6: Checklist de software antes de competir.

Elemento	Verificación
Versión cargada	Confirmada y documentada
Perfil activo	Correcto para la pista o categoría
Parámetros	Revisados y anotados
Calibración	Realizada en condiciones reales
Telemetría	Activada o desactivada según necesidad
Data logger	Verificado si se usará
Seguridad	Parada, borde o protección probada
Modo debug	Desactivado si afecta desempeño

21.16. Pruebas finales por categoría

21.16.1. Seguidor de línea

- Verificar calibración de barra.
- Probar recta, curva suave y curva cerrada.
- Ejecutar al menos tres vueltas válidas.
- Medir mejor tiempo y promedio.
- Revisar pérdida de línea.
- Limpiar llantas antes de ronda.

21.16.2. Minisumo

- Verificar sensores de borde.
- Verificar sensores de oponente.
- Probar arranque y estrategia inicial.
- Revisar empuje y tracción.
- Confirmar peso y dimensiones.
- Revisar batería y fijación del interruptor.

21.16.3. Robot explorador

- Verificar sensores de distancia.
- Probar evasión de obstáculos.
- Revisar comunicación si existe control remoto.
- Validar batería y autonomía.
- Confirmar que los motores no se bloqueen.



Figura 21.6: Pruebas finales por categoría antes de competencia.

21.17. Plan de competencia

Un plan básico debe incluir:

- Robots inscritos y categorías.
- Baterías asignadas por robot.
- Cargadores y adaptadores.
- Repuestos críticos.
- Herramientas.
- Computadora con programas y drivers.
- Versiones finales de código.
- Parámetros impresos o anotados.
- Horario de rondas.
- Responsable de cada robot.

Tabla 21.7: Lista de elementos para llevar a competencia.

Categoría	Elementos
Energía	Baterías, cargadores, extensiones, multímetro
Programación	Computadora, cable USB, drivers, códigos finales
Mecánica	Destornilladores, llaves, tornillos, cinta, pegamento
Electrónica	Cables, conectores, sensores, drivers de repuesto
Limpieza	Alcohol isopropílico, paños, limpiador de llantas
Documentación	Reglamento, parámetros, bitácora, inscripción

21.18. Gestión de riesgos

Antes de competir se deben identificar riesgos:

Tabla 21.8: Matriz básica de riesgos en competencia.

Riesgo	Probabilidad	Impacto	Mitigación
Batería descargada	Media	Alto	Llevar baterías cargadas y medir voltaje
Sensor descalibrado	Alta	Alto	Calibrar antes de ronda
Cable suelto	Media	Alto	Sujetar y revisar conectores
Código incorrecto	Media	Alto	Control de versiones
Llanta sucia	Alta	Medio	Limpiar antes de competir
Driver sobrecalentado	Baja	Alto	Probar consumo y llevar repuesto

21.19. Documentación posterior a la competencia

Después de competir, se debe registrar:

- Resultados obtenidos.
- Fallas presentadas.
- Parámetros usados.
- Condiciones de pista.
- Cambios de último momento.
- Fotos y videos.
- Ideas de mejora.

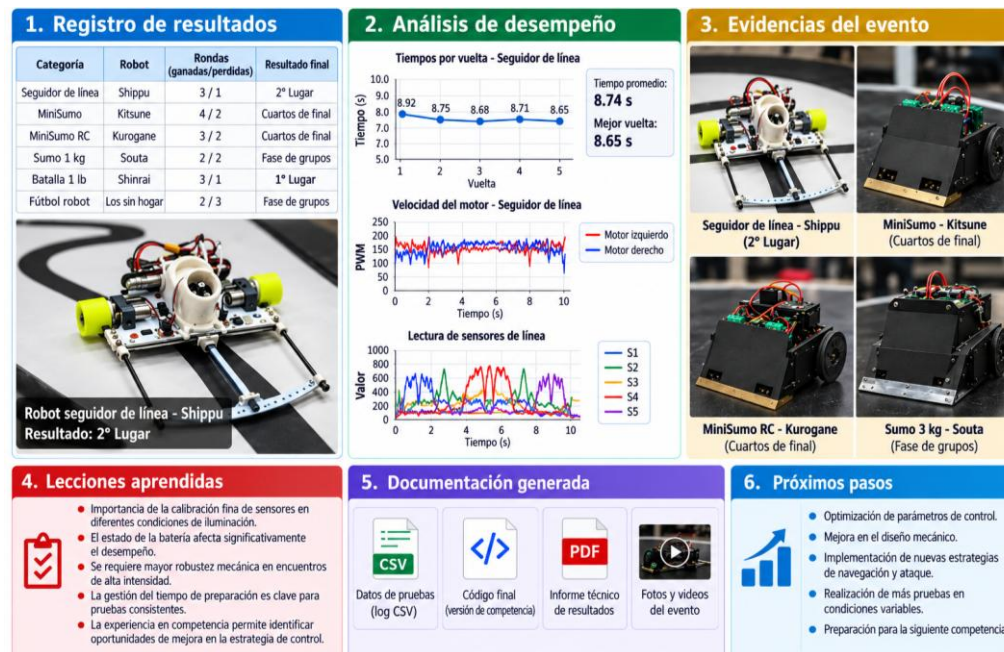


Figura 21.7: Documentación posterior a la competencia.

Esto permite que la experiencia no se pierda y que el siguiente proyecto inicie con mejor información.

21.20. Integración con investigación y docencia

La metodología de integración también puede utilizarse en el aula. Cada robot puede convertirse en un proyecto de aprendizaje basado en evidencias:

- Diseño mecánico.
- Electrónica aplicada.
- Programación embebida.
- Control automático.
- Inteligencia artificial aplicada.
- Análisis de datos.
- Documentación técnica.

En un contexto universitario, esta integración permite relacionar competencias de robótica con resultados de aprendizaje, investigación aplicada y vinculación.

21.21. Buenas prácticas

- Probar por módulos antes de integrar todo.
- Documentar cada cambio importante.
- Mantener una versión estable del programa.
- No hacer cambios mayores justo antes de competir.
- Separar perfiles seguros y agresivos.
- Revisar baterías, sensores y tornillos antes de cada ronda.
- Usar datos para decidir ajustes.
- Registrar aprendizajes después de cada evento.

21.22. Errores frecuentes

- Confiar solo en que el robot funcionó una vez.
- No llevar repuestos críticos.
- Cambiar muchos parámetros antes de una ronda.
- No saber qué programa está cargado.
- No calibrar en la pista real.
- Ignorar pequeños fallos intermitentes.
- No medir voltaje de batería.
- No documentar resultados.

21.23. Proyecto integrador final

Como proyecto final, se propone desarrollar un robot competitivo completo con:

- Control principal documentado.
- Perfiles de competencia.
- Telemetría básica.
- Registro de datos.
- Diagnóstico de fallas.
- Checklist de competencia.
- Bitácora de pruebas.
- Análisis posterior con Python.

El entregable no será únicamente el robot, sino un paquete técnico completo:

- Código fuente.
- Diagrama de conexiones.
- Lista de materiales.
- Parámetros de control.
- Resultados de pruebas.
- Registro de fallas.
- Recomendaciones de mejora.

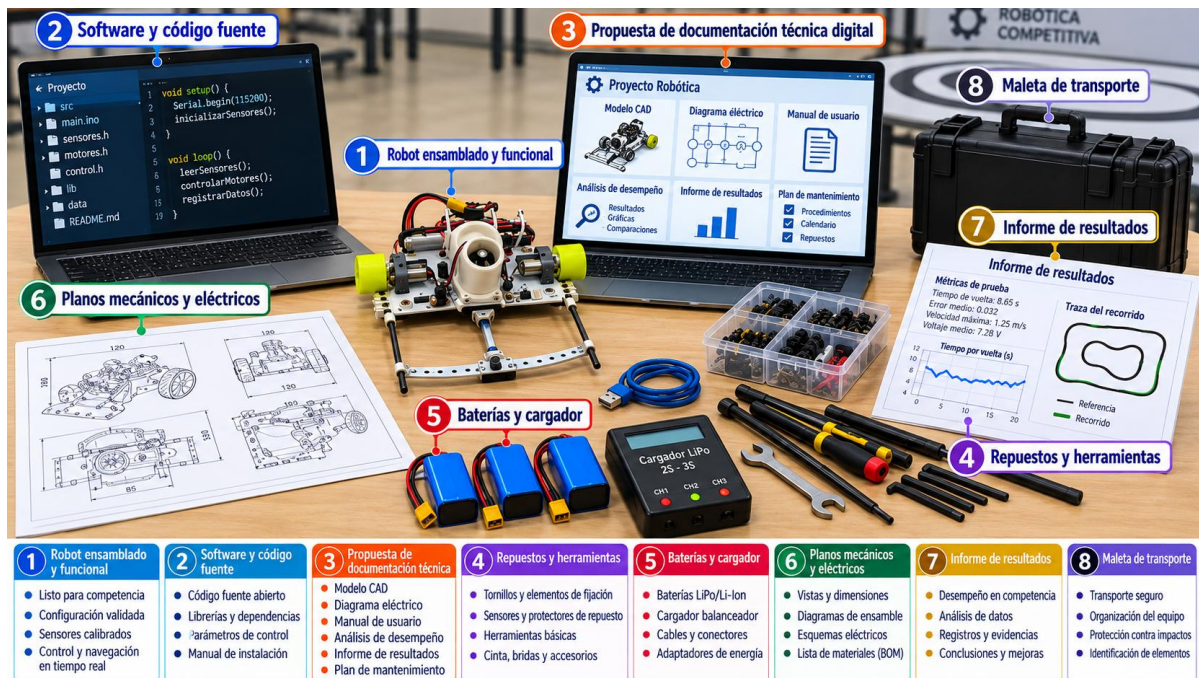


Figura 21.8: Paquete técnico final de un robot competitivo inteligente.

Actividades de aprendizaje

1. Dibuje la arquitectura completa de su robot competitivo.
2. Elabore una lista de módulos de hardware y software.
3. Cree tres perfiles: seguro, rápido y técnico.
4. Complete un checklist mecánico, eléctrico y de software.
5. Diseñe una bitácora de pruebas para su robot.
6. Prepare un plan de competencia con repuestos, baterías, herramientas y versiones de código.

Preguntas de reflexión

1. ¿Por qué un robot rápido puede perder frente a uno más confiable?
2. ¿Qué módulo debe tener mayor prioridad que la estrategia principal?
3. ¿Por qué es peligroso cambiar parámetros justo antes de competir?
4. ¿Qué información debe registrarse en una bitácora de pruebas?
5. ¿Qué repuestos considera indispensables para una competencia?
6. ¿Cómo puede un club de robótica convertir sus competencias en investigación aplicada?

Referencias

- [1] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2017.
- [2] M. Barr and A. Massa, *Programming Embedded Systems: With C and GNU Development Tools*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2006.
- [3] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.
- [4] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*, 2nd ed. Cham, Switzerland: Springer, 2016.
- [5] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ, USA: Princeton University Press, 2008.
- [6] D. C. Montgomery, *Design and Analysis of Experiments*, 10th ed. Hoboken, NJ, USA: Wiley, 2019.
- [7] Arduino, “Arduino Documentation”. [En línea]. Disponible en: <https://docs.arduino.cc/>. [Accedido: 25-jun-2026].
- [8] Espressif Systems, “ESP32 Documentation”. [En línea]. Disponible en: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/>. [Accedido: 25-jun2026].

Glosario

Término	Definición
ADC	Convertidor analógico-digital. Permite que el microcontrolador lea señales analógicas como voltaje de sensores.
Arduino	Plataforma de hardware y software libre ampliamente utilizada para prototipado electrónico y robótica educativa.
Control PID	Controlador proporcional, integral y derivativo usado para corregir errores en sistemas dinámicos.
Control difuso	Técnica de control basada en reglas lingüísticas y grados de pertenencia, útil para decisiones flexibles.
Data logger	Sistema que registra datos durante una prueba, usualmente en memoria microSD o EEPROM.
Dohyo	Área circular de combate utilizada en competencias de robots sumo.
Driver de motor	Circuito de potencia que permite controlar motores desde un microcontrolador.
Encoder	Sensor que mide el giro de un eje o rueda mediante pulsos.
ESP32	Microcontrolador con WiFi y Bluetooth integrado, útil para telemetría, dashboards e IoT.
Fusión sensorial	Combinación de datos de varios sensores para obtener una estimación más robusta.
IMU	Unidad de medición inercial. Generalmente integra acelerómetro y giroscopio.
LiPo	Batería de polímero de litio, común en robótica por su alta densidad de energía y capacidad de descarga.
MicroSD	Tarjeta de memoria usada para almacenar datos, registros o archivos CSV.
Minisumo	Categoría de robot de combate autónomo o controlado, orientada a sacar al oponente del dohyo.
Odometría	Estimación del movimiento de un robot a partir del giro de sus ruedas.
PWM	Modulación por ancho de pulso. Técnica utilizada para controlar velocidad de motores o intensidad de señales.
Sensor reflectivo	Sensor que mide luz reflejada, comúnmente usado para detectar líneas o bordes.
TB6612FNG	Driver de motores DC de doble canal, usado en robots móviles pequeños y medianos.
Telemetría	Envío de datos del robot hacia una computadora, celular o dashboard para monitoreo.
TinyML	Implementación de modelos de aprendizaje automático en microcontroladores de bajo recurso.



“De la idea
a la solución real,
la tecnología se construye
con conocimiento,
creatividad y propósito.”




EDITORIAL
SAGA

ISBN: 978-9907-803-72-3



9 789907 803723