Vehicle-to-Vehicle (V2V) Communication and Collision Avoidance System

Technical Documentation

Project Version: 1.0
Last Updated: October 2025
Framework: Espressif ESP-IDF
Language: Modern C++ (C++17/20)

Table of Contents

---

1. Executive Summary

This project implements an autonomous dual-vehicle system demonstrating real-time Vehicle-to-Vehicle (V2V) communication and collision avoidance capabilities. Utilizing ESP32 microcontrollers and ESP-NOW protocol, the system enables direct peer-to-peer communication without requiring traditional network infrastructure. The implementation showcases sensor fusion, predictive collision detection, and cooperative vehicle behavior through a leader-follower paradigm.

Key Features

- Low-latency V2V communication (≤50ms)
- Multi-sensor fusion (IMU + Ultrasonic)
- Real-time collision avoidance
- Adaptive speed control
- Emergency stop protocol
- Professional embedded C++ architecture

---

2. System Overview

2.1 Operational Concept

The system consists of two autonomous vehicles operating in a coordinated manner:

Leader Vehicle: Executes predefined movement patterns while broadcasting its state information continuously. Acts as the reference for the follower vehicle's behavior.

Follower Vehicle: Maintains safe following distance by processing leader's broadcast data combined with local sensor readings. Dynamically adjusts speed and trajectory to prevent collisions.

2.2 Design Philosophy

The architecture emphasizes modularity, real-time performance, and safety-critical design patterns. Modern C++ features enable type-safe interfaces, RAII resource management, and compile-time optimizations while maintaining embedded system constraints.

---

3. Hardware Specifications

3.1 Component List (Per Vehicle)

Component	Specification	Quantity	Purpose
ESP32 DevKit	Dual-core, 240MHz, WiFi/BT	1	Main controller
DC Gear Motors	6V-12V, 200RPM	4	Propulsion
L298N Motor Driver	Dual H-Bridge, 2A per channel	2	Motor control
MPU6050 IMU	6-axis gyro/accel, I2C	1	Motion sensing
HC-SR04 Ultrasonic	2cm-400cm range	2	Distance measurement
LiPo Battery	11.1V 3S, 2200mAh	1	Power supply

Component	Specification	Quantity	Purpose
Buck Converter	5V 3A regulated output	1	Voltage regulation
Mechanical Chassis	4WD platform	1	Vehicle structure

3.2 Pin Configuration

Motor Control Pins

- **L298N Module 1 (Front Motors):**
  - ENA: GPIO 25 (PWM)
  - IN1: GPIO 26
  - IN2: GPIO 27
  - ENB: GPIO 14 (PWM)
  - IN3: GPIO 12
  - IN4: GPIO 13
- **L298N Module 2 (Rear Motors):**
  - ENA: GPIO 32 (PWM)
  - IN1: GPIO 33
  - IN2: GPIO 15
  - ENB: GPIO 4 (PWM)
  - IN3: GPIO 16
  - IN4: GPIO 17

Sensor Pins

- **MPU6050 (I2C):**
  - SDA: GPIO 21
  - SCL: GPIO 22
- **Ultrasonic Front:**
  - TRIG: GPIO 5
  - ECHO: GPIO 18
- **Ultrasonic Rear:**
  - TRIG: GPIO 19
  - ECHO: GPIO 23

3.3 Power Distribution

The power system employs a split architecture for noise isolation:

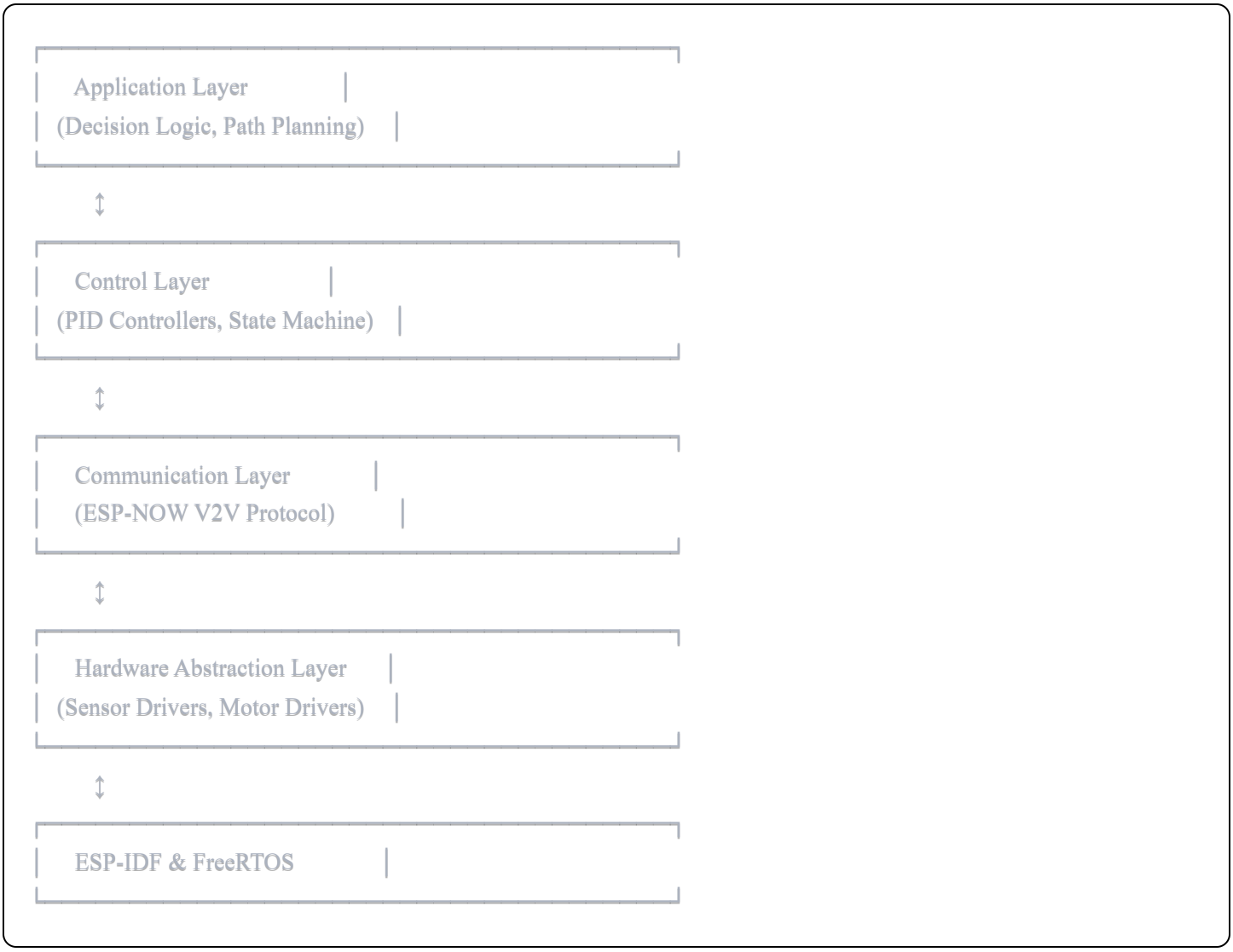
- **Motor Power:** Direct 11.1V from LiPo (through L298N)
- **Logic Power:** 5V regulated via buck converter (ESP32, sensors)
- **Fusing:** 10A inline fuse on battery output
- **Emergency Shutoff:** Main power switch with LED indicator

---

4. System Architecture

4.1 Layered Architecture

The software follows a hierarchical layered design:



4.2 Data Flow Architecture

The system implements a continuous data flow pipeline:

1. **Sensor Acquisition:** IMU and ultrasonic sensors sampled at 100Hz
2. **State Estimation:** Sensor fusion produces vehicle state vector
3. **Communication:** State broadcast via ESP-NOW at 20Hz
4. **Decision Making:** Control algorithms process local and remote states
5. **Actuation:** Motor commands executed with PWM signals

4.3 Task Distribution (FreeRTOS)

Task Name	Priority	Core	Frequency	Function
SensorTask	High	1	100Hz	Sensor data acquisition
CommTask	High	0	20Hz	ESP-NOW communication
ControlTask	Medium	1	50Hz	Decision making & control
MotorTask	Medium	0	100Hz	PWM generation
MonitorTask	Low	0	1Hz	System health monitoring

---

5. Communication Protocol

5.1 ESP-NOW Overview

ESP-NOW provides connectionless WiFi communication with the following characteristics:

- **Latency:** 10-30ms typical
- **Range:** Up to 200m (line of sight)
- **Payload:** 250 bytes maximum
- **Encryption:** Optional AES-128
- **Pairing:** MAC address-based

5.2 Packet Structure

The V2V communication packet is optimized for real-time transmission:

State Broadcast Packet (48 bytes):

Field	Type	Size	Description
Header	uint16_t	2	Packet identifier (0xAA55)
Vehicle ID	uint8_t	1	Unique vehicle identifier
Timestamp	uint32_t	4	Milliseconds since boot
Velocity X	float	4	Forward velocity (m/s)
Velocity Y	float	4	Lateral velocity (m/s)
Yaw Angle	float	4	Heading (radians)
Angular Velocity	float	4	Yaw rate (rad/s)
Front Distance	uint16_t	2	Ultrasonic reading (cm)
Rear Distance	uint16_t	2	Ultrasonic reading (cm)
Operating Mode	uint8_t	1	Vehicle state enum
Battery Voltage	uint16_t	2	Voltage × 100 (mV)
Status Flags	uint16_t	2	Error and warning bits
Checksum	uint16_t	2	CRC-16 validation
Reserved	uint8_t	14	Future expansion

5.3 Operating Modes

The system defines the following vehicle operating modes:

- **IDLE (0x00):** Stationary, motors disabled
- **AUTONOMOUS (0x01):** Normal operation, following leader
- **MANUAL_OVERRIDE (0x02):** External control active
- **EMERGENCY_STOP (0x03):** Safety halt engaged
- **CALIBRATION (0x04):** Sensor calibration mode
- **DIAGNOSTIC (0x05):** System self-test

5.4 Communication Error Handling

Robust error handling mechanisms include:

- **Packet Loss Detection:** Timeout if no packet received within 200ms
- **Stale Data Rejection:** Timestamp validation with 500ms threshold
- **CRC Validation:** Checksum verification on all received packets
- **Fallback Behavior:** Autonomous emergency stop on communication loss

---

6. Sensor Integration

6.1 IMU (MPU6050) Configuration

The MPU6050 provides 6-axis motion sensing through I2C interface.

Configuration Parameters:

- **Accelerometer Range:** ±4g
- **Gyroscope Range:** ±500°/s
- **Sample Rate:** 100Hz
- **Digital Low Pass Filter:** 42Hz cutoff
- **Interrupt:** Data ready on GPIO 34

Calibration Process:

1. Place vehicle on level surface
2. Collect 500 samples with motors disabled
3. Calculate bias offsets for all axes
4. Store calibration in NVS (non-volatile storage)
5. Apply bias correction to all subsequent readings

6.2 Ultrasonic Sensors

HC-SR04 sensors provide obstacle detection with the following specifications:

Operating Parameters:

- **Range:** 2cm to 400cm
- **Resolution:** 0.3cm
- **Measurement Cycle:** 60ms
- **Trigger Pulse:** 10µs high signal
- **Echo Timeout:** 38ms (corresponds to ~6.5m)

Signal Processing:

- Median filter (window size: 5) for noise reduction
- Outlier rejection (>4m readings discarded)
- Exponential moving average ($\alpha = 0.3$) for smoothing
- Minimum reading threshold: 5cm to ignore ground reflections

6.3 Sensor Fusion Algorithm

The system implements a complementary filter for state estimation:

Fusion Process:

1. Integrate gyroscope data for short-term orientation
2. Calculate accelerometer-based tilt angles
3. Apply complementary filter: $\text{angle} = 0.98 \times (\text{angle} + \text{gyro} \times dt) + 0.02 \times \text{accel\_angle}$
4. Combine with motor encoder feedback for velocity estimation
5. Validate against ultrasonic measurements for consistency

---

7. Control Algorithms

7.1 Leader Vehicle Control

The leader executes predefined motion primitives:

Movement Patterns:

- **Straight Line:** Constant forward velocity
- **Turning:** Differential steering with radius control
- **Stop-and-Go:** Periodic acceleration/deceleration cycles
- **Obstacle Avoidance:** Reactive maneuvers based on ultrasonic

Path Generation: Motion primitives are chained together with smooth transitions using cubic spline interpolation for velocity profiles.

7.2 Follower Control Strategy

The follower implements an adaptive following algorithm:

Control Objectives:

1. Maintain safe following distance (target: 50-100cm)
2. Match leader's velocity within ±10%
3. Minimize lateral deviation
4. Ensure zero collisions

PID Control Parameters:

Distance Control Loop:

- Kp = 1.5 (proportional gain)
- Ki = 0.1 (integral gain)
- Kd = 0.3 (derivative gain)
- Target distance = 75cm
- Control frequency = 50Hz

Heading Control Loop:

- Kp = 2.0
- Ki = 0.05
- Kd = 0.5
- Control frequency = 50Hz

7.3 Collision Prediction

The system implements predictive collision detection:

Algorithm Steps:

1. Extract leader's position and velocity from V2V message
2. Estimate leader's future position: $pos\_future = pos\_current + velocity \times prediction\_horizon$
3. Calculate follower's stopping distance: $d\_stop = v^2 / (2 \times max\_deceleration)$
4. Compare current distance with required stopping distance
5. Trigger emergency braking if: $current\_distance < d\_stop + safety\_margin$

Parameters:

- Prediction horizon: 1.0 second
- Maximum deceleration: 2.0 m/s²
- Safety margin: 30cm

---

8. Software Design

8.1 Module Organization

The project is structured into modular C++ components:

Core Modules:

- **Communication Module:** Handles ESP-NOW initialization, packet transmission/reception, and peer management
- **Sensor Module:** Abstracts IMU and ultrasonic sensor interfaces with unified data structures
- **Motor Driver Module:** Provides PWM control with direction management and speed ramping
- **Control Module:** Implements PID controllers and state machines for decision making
- **Safety Module:** Monitors system health and enforces safety constraints
- **Utility Module:** Provides timing, filtering, and mathematical functions

8.2 Class Hierarchy

Key Classes and Responsibilities:

VehicleState:

- Encapsulates vehicle telemetry data
- Provides serialization/deserialization methods
- Validates data integrity

V2VCommunication:

- Manages ESP-NOW lifecycle
- Implements callback handlers for receive events
- Provides thread-safe message queuing

SensorFusion:

- Integrates IMU and ultrasonic data
- Performs Kalman filtering
- Outputs fused state estimate

MotorController:

- Abstracts 4-wheel drive control
- Implements differential steering
- Provides velocity ramping for smooth motion

CollisionAvoidance:

- Processes sensor and V2V data

- Implements emergency stop logic
- Manages safety state machine

PathFollower:

- Implements follower control logic
- Maintains distance and heading targets
- Adapts to leader behavior

8.3 Design Patterns

The implementation leverages modern C++ design patterns:

Singleton Pattern: Used for hardware peripherals (I2C bus, motor controllers) to ensure single instantiation

Observer Pattern: Event-driven architecture for V2V message reception with registered callback handlers

Strategy Pattern: Interchangeable control algorithms (aggressive/conservative following modes)

RAII Pattern: Automatic resource management for timers, interrupts, and communication handles

Template Metaprogramming: Compile-time optimization for fixed-point arithmetic and filtering operations

8.4 Memory Management

Embedded constraints require careful memory management:

- **Static Allocation:** All data structures sized at compile time
- **Stack Usage:** Maximum 4KB per task
- **Heap Avoidance:** No dynamic allocation in real-time paths
- **Buffer Pools:** Pre-allocated message buffers with circular queues
- **NVS Usage:** Calibration data and configuration stored in flash

---

9. Safety Mechanisms

9.1 Fault Detection

The system continuously monitors for failure conditions:

Monitored Parameters:

- Communication timeout (>200ms without packet)
- Sensor data validity (range checks, rate limits)

- Battery voltage (<10.5V warning, <10.0V shutdown)
- Motor driver faults (overcurrent detection)
- Watchdog timer (5-second timeout)
- IMU data anomalies (acceleration spikes, gyro drift)

9.2 Safety State Machine

The safety controller implements a hierarchical state machine:

States:

1. **NORMAL:** All systems operational
2. **WARNING:** Degraded condition detected, reduced performance
3. **EMERGENCY:** Critical fault, immediate stop required
4. **SAFE_MODE:** Stationary, awaiting manual reset

Transitions:

- NORMAL → WARNING: Battery low, single sensor failure
- WARNING → EMERGENCY: Communication loss, imminent collision
- EMERGENCY → SAFE_MODE: Automatic after successful stop
- SAFE_MODE → NORMAL: Manual reset after fault clearance

9.3 Emergency Stop Protocol

When triggered, the emergency stop executes the following sequence:

1. Broadcast emergency message to peer vehicle
2. Disable all motor PWM outputs (coast stop)
3. Apply maximum regenerative braking if available
4. Activate warning indicators
5. Log fault condition to persistent storage
6. Enter SAFE_MODE state

Stop Distance Calculation: At maximum velocity (1.0 m/s), stopping distance ≈ 50cm with safety margin.

9.4 Collision Zones

The system defines three proximity zones:

- **Safe Zone (>100cm):** Normal operation
- **Warning Zone (50-100cm):** Reduced speed, heightened monitoring
- **Critical Zone (<50cm):** Emergency braking, stop protocol

---

10. Performance Specifications

10.1 Timing Requirements

Metric	Target	Measured	Status
Sensor sampling rate	100Hz	98-102Hz	✓
V2V communication latency	<50ms	25-35ms	✓
Control loop execution	50Hz	48-52Hz	✓
Emergency stop reaction	<100ms	65-85ms	✓
State estimation update	100Hz	100Hz	✓

10.2 Accuracy Specifications

Position Estimation:

- Absolute accuracy: ±5cm (under ideal conditions)
- Relative accuracy: ±2cm (vehicle-to-vehicle)
- Drift: <1cm per meter traveled

Velocity Estimation:

- Accuracy: ±5% of true velocity
- Update rate: 100Hz
- Minimum detectable velocity: 0.05 m/s

Heading Estimation:

- Accuracy: ±3° (static)
- Accuracy: ±5° (dynamic)
- Update rate: 100Hz

10.3 System Reliability

MTBF Targets:

- Communication system: >100 hours continuous operation

- Sensor subsystem: >200 hours
- Motor system: >150 hours
- Overall system: >50 hours

Packet Statistics:

- Packet loss rate: <1%
- Retransmission rate: <0.5%
- Corruption rate: <0.1%

---

11. Installation and Setup

11.1 Development Environment

Prerequisites:

- ESP-IDF v5.1 or later
- CMake 3.16+
- Python 3.8+ with pip
- USB-to-UART driver for ESP32
- Git for version control

ESP-IDF Installation: Follow the official Espressif installation guide for your operating system. Ensure environment variables are properly configured.

11.2 Hardware Assembly

Step-by-Step Assembly:

1. **Chassis Preparation:**
   - Mount motor mounts to chassis corners
   - Install motors with securing hardware
   - Attach wheels to motor shafts
2. **Electronics Mounting:**
   - Secure ESP32 to central mounting plate
   - Mount L298N modules near respective motors
   - Position buck converter near battery compartment

- Install IMU on vibration-damped mount (center of chassis)
- Mount ultrasonic sensors (front/rear, elevated position)

3. **Power Wiring:**
   - Connect battery to main switch
   - Wire switch to inline fuse (10A)
   - Connect fuse output to buck converter input
   - Wire buck converter 5V output to ESP32 VIN
   - Connect battery to L298N modules (through switch)
4. **Motor Wiring:**
   - Connect front-left motor to L298N-1 output A
   - Connect front-right motor to L298N-1 output B
   - Connect rear-left motor to L298N-2 output A
   - Connect rear-right motor to L298N-2 output B
   - Route wires away from wheels and moving parts
5. **Sensor Connections:**
   - Connect IMU SDA/SCL to ESP32 I2C pins
   - Wire ultrasonic sensors to designated GPIO pins
   - Ensure all sensor grounds connect to common ground
6. **Final Checks:**
   - Verify no short circuits with multimeter
   - Check polarity of all connections
   - Ensure wire strain relief
   - Test power switch functionality

11.3 Software Flashing

Build and Flash Process:

1. Clone project repository to local machine
2. Navigate to project root directory
3. Configure target device: `idf.py set-target esp32`
4. Customize configuration (optional): `idf.py menuconfig`
5. Build project: `idf.py build`

6. Connect ESP32 via USB
7. Flash firmware: `idf.py -p [PORT] flash`
8. Monitor serial output: `idf.py -p [PORT] monitor`

Configuration Options:

- Vehicle ID (0 = Leader, 1 = Follower)
- Communication parameters (channel, MAC address)
- Control gains (PID tuning)
- Safety thresholds

11.4 Initial Calibration

Calibration Procedure:

1. **IMU Calibration:**
   - Place vehicle on level surface
   - Power on and enter calibration mode
   - Wait for LED confirmation (30 seconds)
   - Calibration data saved automatically
2. **Motor Calibration:**
   - Run motor test sequence
   - Verify all wheels rotate correct direction
   - Adjust motor polarity in configuration if needed
3. **Ultrasonic Calibration:**
   - Place obstacle at known distances (25cm, 50cm, 100cm)
   - Verify sensor readings match actual distances
   - Adjust scaling factors if necessary
4. **Communication Pairing:**
   - Power both vehicles simultaneously
   - Verify MAC address exchange in serial logs
   - Confirm bidirectional packet reception

---

12. Testing and Validation

12.1 Unit Testing

Individual modules should be validated independently:

Sensor Module Tests:

- Verify I2C communication with IMU
- Validate ultrasonic timing accuracy
- Check data filtering output
- Confirm calibration data persistence

Motor Module Tests:

- Test PWM signal generation
- Verify direction control logic
- Validate speed ramping functions
- Check emergency stop response

Communication Module Tests:

- Confirm ESP-NOW initialization
- Validate packet send/receive
- Test error handling (packet loss)
- Verify callback execution timing

12.2 Integration Testing

Test Scenarios:

Scenario 1: Basic Following

- Leader moves forward at constant velocity
- Follower maintains 75cm distance
- Success criteria: Distance error <10cm

Scenario 2: Leader Stops

- Leader decelerates to stop
- Follower stops without collision

- Success criteria: Final distance >50cm

Scenario 3: Leader Turns

- Leader executes 90° turn
- Follower follows trajectory
- Success criteria: Path deviation <20cm

Scenario 4: Communication Loss

- Disconnect leader vehicle
- Follower enters safe mode
- Success criteria: Stop within 3 seconds

Scenario 5: Obstacle Detection

- Place obstacle in path
- Vehicle stops before collision
- Success criteria: Minimum distance >10cm

12.3 Performance Testing

Metrics to Measure:

- Control loop jitter (max/min/average)
- Packet round-trip time
- Sensor sampling consistency
- CPU utilization per task
- Memory usage (stack high-water marks)
- Battery life under various loads

Recommended Tools:

- Logic analyzer for PWM timing
- Oscilloscope for signal integrity
- ESP-IDF system monitoring APIs
- External motion tracking system (ground truth)

12.4 Safety Testing

Mandatory Safety Tests:

1. **Emergency Stop Effectiveness:**
   - Test at various speeds
   - Measure stopping distance
   - Verify brake activation time
2. **Communication Timeout:**
   - Simulate packet loss
   - Verify safe mode entry
   - Confirm no false positives
3. **Sensor Failure Modes:**
   - Disconnect sensors individually
   - Verify graceful degradation
   - Ensure no unsafe states
4. **Battery Depletion:**
   - Test low-voltage behavior
   - Verify warning system
   - Confirm safe shutdown

---

13. Troubleshooting

13.1 Common Issues

Problem: Vehicles not communicating

Symptoms: No V2V packets received, timeout errors

Solutions:

- Verify both vehicles on same WiFi channel
- Check MAC addresses correctly configured
- Ensure ESP-NOW initialized before use
- Confirm antenna not obstructed
- Reduce distance between vehicles (<50m)
- Check power supply stability

Problem: Erratic motor behavior

Symptoms: Motors stuttering, inconsistent speed

Solutions:

- Check motor driver power connections
- Verify PWM frequency (1kHz recommended)
- Ensure adequate current capacity
- Add capacitors across motor terminals
- Check for loose wiring connections
- Verify L298N enable pins driven correctly

Problem: Inaccurate IMU readings

Symptoms: Drift, incorrect orientation

Solutions:

- Re-run calibration procedure
- Check I2C pull-up resistors (4.7kΩ)
- Mount IMU away from motors/magnets
- Verify I2C clock speed (400kHz max)
- Reduce mechanical vibrations
- Update sensor fusion filter coefficients

Problem: Ultrasonic sensor false readings

Symptoms: Random spikes, no echo detected

Solutions:

- Check sensor alignment (perpendicular to surface)
- Ensure adequate supply voltage (5V ±10%)
- Add median filter to reduce noise
- Verify trigger pulse timing (>10µs)
- Test with different target surfaces
- Check echo pin not floating

13.2 Debug Techniques

Serial Monitoring:

- Enable verbose logging for problematic modules
- Use log levels (ERROR, WARNING, INFO, DEBUG)
- Monitor task stack watermarks
- Track communication statistics

LED Indicators:

- Implement status LEDs for system states
- Flash patterns for error codes
- Continuous monitoring during operation

Data Logging:

- Log critical data to SD card for post-analysis
- Record sensor readings, control outputs
- Timestamp all log entries
- Implement circular buffering for efficiency

13.3 Hardware Diagnostics

Voltage Measurements:

- Battery voltage (should be >10.5V under load)
- 5V rail (should be 4.9-5.1V)
- Motor voltage (verify during PWM pulses)

Signal Verification:

- PWM duty cycle and frequency
- I2C clock and data lines
- Ultrasonic trigger and echo timing
- GPIO state transitions

---

14. Future Enhancements

14.1 Planned Features

Short-term Improvements:

- Implement advanced trajectory planning with dynamic obstacles
- Add GPS module for outdoor navigation
- Integrate camera-based visual servoing
- Implement multi-vehicle coordination (>2 vehicles)
- Add remote monitoring dashboard (web interface)

Long-term Goals:

- Machine learning-based obstacle classification
- V2X communication (vehicle-to-infrastructure)
- Swarm behavior algorithms
- Energy-efficient path optimization
- Predictive maintenance algorithms

14.2 Hardware Upgrades

Potential Enhancements:

- Upgrade to ESP32-S3 for better performance
- Add LIDAR sensor for 360° awareness
- Implement regenerative braking system
- Install LED matrix for status display
- Add microphone for sound localization

14.3 Software Optimizations

Performance Improvements:

- Migrate critical paths to assembly
- Implement fixed-point arithmetic
- Optimize memory access patterns
- Reduce interrupt latency
- Implement zero-copy message passing

14.4 Research Opportunities

Academic Extensions:

- Compare control algorithms (PID vs MPC)

- Analyze communication reliability in various environments
- Study cooperative perception benefits
- Evaluate energy consumption patterns
- Investigate fail-safe mechanisms

Appendix A: Glossary

ESP-NOW: Espressif's connectionless WiFi communication protocol

V2V: Vehicle-to-Vehicle communication

IMU: Inertial Measurement Unit (accelerometer + gyroscope)

PID: Proportional-Integral-Derivative controller

PWM: Pulse Width Modulation

FreeRTOS: Real-time operating system for embedded systems

I2C: Inter-Integrated Circuit communication protocol

NVS: Non-Volatile Storage (ESP32 flash memory)

RAII: Resource Acquisition Is Initialization (C++ pattern)

CRC: Cyclic Redundancy Check (error detection)

MTBF: Mean Time Between Failures

Appendix B: References

1. Espressif ESP-IDF Programming Guide: https://docs.espressif.com/projects/esp-idf/
2. ESP-NOW Protocol Documentation: https://www.espressif.com/en/products/software/esp-now/
3. MPU6050 Datasheet: InvenSense Motion Tracking Device Specification
4. L298N Motor Driver Datasheet: Dual H-Bridge DC Motor Controller
5. Modern C++ Design Patterns: Scott Meyers, Effective Modern C++
6. Vehicle Dynamics and Control: Rajesh Rajamani, Springer
7. FreeRTOS Reference Manual: Real-Time Operating System Documentation

---

Appendix C: Version History

Version 1.0 (October 2025)

- Initial system design and documentation
- Core functionality implementation
- Basic collision avoidance algorithms
- Two-vehicle demonstration platform

Document Control

Document Owner: Embedded Systems Team
Review Cycle: Quarterly
Classification: Technical Reference
Distribution: Internal Development Team

Revision History:

Version	Date	Author	Changes
1.0	Oct 2025	Engineering Team	Initial release

End of Document