Design and Implementation of a Wi-Fi BOT with Real-time Camera Feed

Project Report

Executive Summary

This project presents the development of a Wi-Fi controlled robot car equipped with a live camera feed using an ESP32 microcontroller. The system enables remote operation through a web-based interface, allowing users to control vehicle movement, adjust speed, monitor the environment through real-time video streaming, and control an external light source. The implementation leverages asynchronous web servers and WebSocket technology to achieve low-latency control and smooth video transmission.

1. Introduction

1.1 Project Overview

The Wi-Fi Camera Car Control System is an embedded IoT project that integrates robotics, wireless communication, and real-time video streaming. The system is designed for applications in surveillance, remote exploration, and educational demonstrations of IoT principles.

1.2 Objectives

- Develop a remotely controllable robot car using an ESP32 microcontroller
- Implement real-time video streaming over Wi-Fi
- Create an intuitive web-based control interface
- Enable adjustable speed control and external lighting
- Establish reliable bidirectional communication using WebSockets

1.3 Scope

The project encompasses hardware integration, firmware development, web interface design, and real-time communication protocol implementation. It demonstrates practical applications of embedded systems, wireless networking, and IoT technologies.

---

2. System Architecture

2.1 Hardware Components

Microcontroller:

- ESP32 development board with built-in Wi-Fi capabilities
- Dual-core processor running at 240 MHz
- PSRAM support for efficient memory management

Camera Module:

- ESP32-compatible camera module
- VGA resolution (640x480) support
- JPEG compression for efficient transmission

Motor System:

- Two DC motors for differential drive
- L298N or similar motor driver module
- PWM-based speed control

Additional Components:

- External LED light (GPIO 4)
- Power supply system
- Chassis and mechanical components

2.2 Pin Configuration

Motor Connections:

- Right Motor: Enable (GPIO 12), IN1 (GPIO 13), IN2 (GPIO 15)
- Left Motor: Enable (GPIO 12), IN3 (GPIO 14), IN4 (GPIO 2)

Camera Interface:

- Data pins: GPIO 5, 18, 19, 21, 36, 39, 34, 35
- Control pins: XCLK (GPIO 0), PCLK (GPIO 22), VSYNC (GPIO 25), HREF (GPIO 23)
- I2C: SDA (GPIO 26), SCL (GPIO 27)
- Power control: PWDN (GPIO 32)

Light Control:

- LED: GPIO 4

2.3 Software Architecture

The system employs a three-tier architecture:

Firmware Layer (ESP32):

- Motor control logic
- Camera frame acquisition and streaming
- WebSocket server management
- PWM signal generation
- Event handling

Communication Layer:

- AsyncTCP for non-blocking network operations
- WebSocket protocol for real-time bidirectional communication
- HTTP server for web interface delivery

Presentation Layer:

- HTML5/CSS3 responsive web interface
- JavaScript for client-side logic
- WebSocket client implementation

---

3. Implementation Details

3.1 Wi-Fi Configuration

The ESP32 operates in Access Point (AP) mode, creating its own Wi-Fi network with the following credentials:

- SSID: "HawkEye"

- Password: "12345678"
- Default IP: 192.168.4.1

This configuration allows direct connection without requiring an existing network infrastructure, making the system portable and independent.

3.2 Motor Control System

Movement Logic: The system implements differential drive kinematics:

- Forward: Both motors rotate forward
- Backward: Both motors rotate backward
- Left Turn: Right motor forward, left motor backward
- Right Turn: Right motor backward, left motor forward
- Stop: Both motors disabled

PWM Configuration:

- Frequency: 1 kHz
- Resolution: 8-bit (0-255 speed range)
- Channel 2: Motor speed control
- Channel 3: Light intensity control

3.3 Camera System

Initialization Parameters:

- Frame size: VGA (640x480)
- JPEG quality: 10 (lower value = higher quality)
- Frame buffer count: 1
- Pixel format: JPEG
- Clock frequency: 20 MHz

Streaming Process:

1. Capture frame using `esp_camera_fb_get()`
2. Transmit binary data via WebSocket
3. Return frame buffer to pool
4. Monitor queue status to prevent overflow
5. Repeat continuously in main loop

3.4 WebSocket Communication

Two Independent Channels:

Camera WebSocket (/Camera):

- Handles video stream transmission
- Binary data transfer for image frames
- Tracks connected client ID
- Automatic reconnection on disconnect

Control WebSocket (/CarInput):

- Processes movement commands
- Handles speed and light adjustments
- Text-based command protocol: "Key,Value"
- Bidirectional communication support

3.5 Web Interface Design

User Interface Features:

- Touch-optimized directional controls
- Real-time camera feed display
- Speed control slider (0-255)
- Light intensity slider (0-255)
- Responsive design for mobile devices
- Visual feedback on button press

Technical Implementation:

- HTML5 Canvas for video rendering
- CSS3 for styling and animations
- Vanilla JavaScript for logic
- WebSocket API for communication
- Event listeners for touch interactions

4. Functional Workflow

4.1 System Startup Sequence

1. ESP32 initializes serial communication (115200 baud)

2. GPIO pins configured for motors and light
3. PWM channels set up for speed and light control
4. Wi-Fi Access Point started
5. HTTP server initialized on port 80
6. WebSocket handlers registered
7. Camera module initialized and configured
8. System enters main operation loop

4.2 Operation Cycle

1. Client connects to ESP32 Wi-Fi network
2. Browser accesses web interface (http://192.168.4.1)
3. Two WebSocket connections established
4. Initial speed and light values transmitted
5. Camera begins continuous frame capture
6. User interactions trigger command transmission
7. ESP32 processes commands and updates outputs
8. Camera frames streamed to client continuously

4.3 Command Processing Flow

1. User interaction (button press/slider change)
2. JavaScript generates command string
3. WebSocket sends command to ESP32
4. ESP32 parses command (key-value pair)
5. Appropriate handler executes action
6. Physical output updated (motors/light)
7. System ready for next command

5. Technical Challenges and Solutions

5.1 Memory Management

Challenge: Camera frames consume significant memory, potentially causing heap fragmentation.

Solution:

- Utilized PSRAM (external RAM) for frame buffer allocation

- Implemented immediate frame buffer return after transmission
- Single frame buffer configuration to minimize memory usage
- Continuous monitoring of available heap memory

5.2 Network Latency

Challenge: Ensuring responsive control despite wireless communication delays.

Solution:

- Asynchronous TCP implementation for non-blocking operations
- WebSocket protocol for low-latency bidirectional communication
- Client-side command buffering
- Queue management to prevent overflow

5.3 Video Streaming Quality

Challenge: Balancing image quality, frame rate, and bandwidth.

Solution:

- JPEG compression to reduce data size
- VGA resolution as optimal balance
- Queue monitoring to prevent frame accumulation
- Adjustable JPEG quality parameter

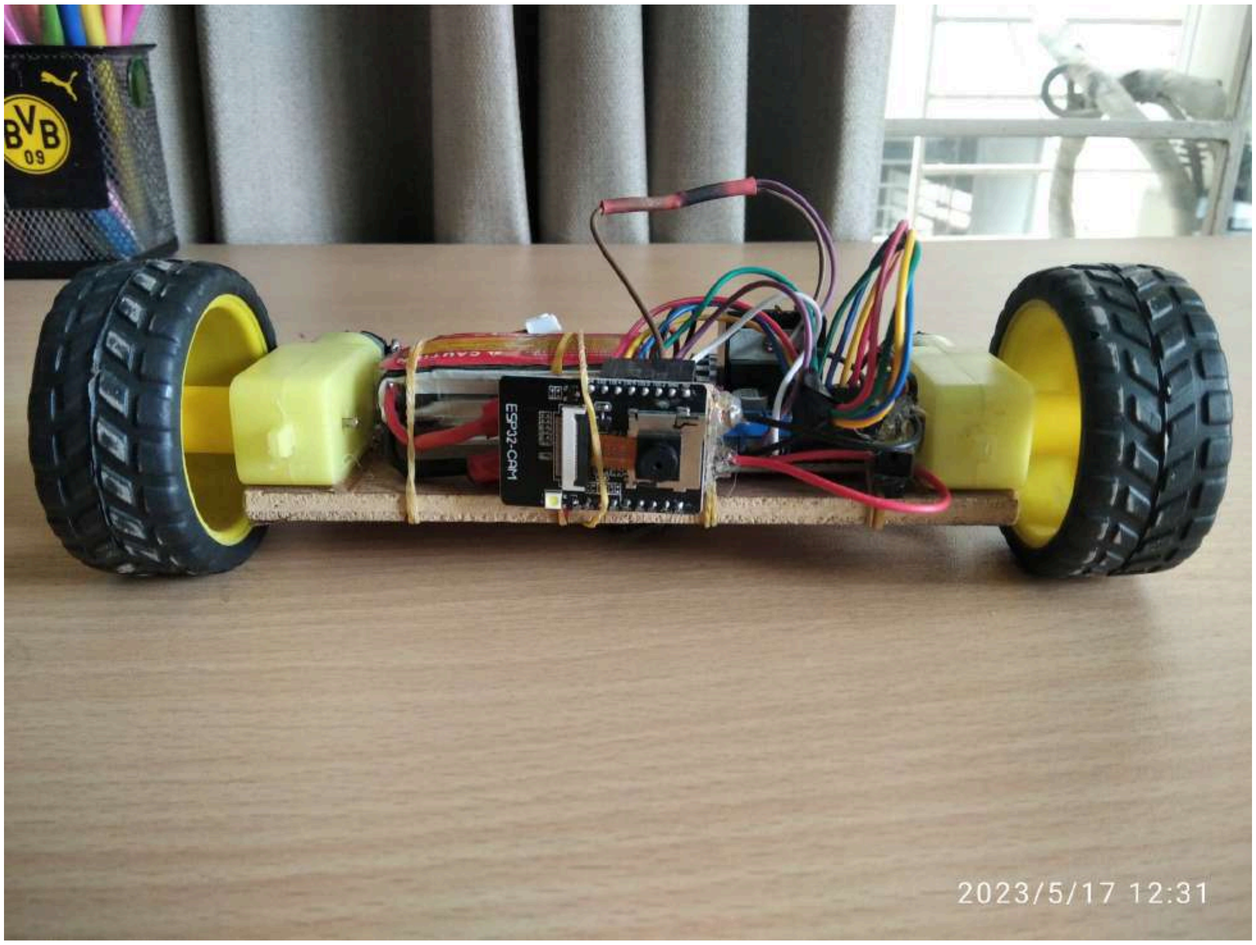
5.4 Connection Stability

Challenge: Maintaining stable WebSocket connections.

Solution:

- Automatic reconnection logic on client side
- Connection state monitoring
- Graceful handling of disconnections
- Safety mechanism to stop motors on disconnect

2023/5/17 12:31



6. Performance Analysis

6.1 Key Metrics

Video Performance:

- Frame rate: Approximately 10-20 FPS (dependent on network conditions)
- Resolution: 640x480 pixels
- Average frame transmission time: 50-100ms
- JPEG compression ratio: Approximately 1:10

Control Responsiveness:

- Command latency: <50ms under normal conditions
- WebSocket connection establishment: <500ms
- Motor response time: Immediate upon command reception

Network Performance:

- Wi-Fi range: Typical 20-30 meters (indoor)
- Maximum concurrent connections: 4 clients
- Bandwidth utilization: Approximately 1-2 Mbps

6.2 System Limitations

- Single active camera stream support
- Limited range due to Wi-Fi constraints
- Battery life dependent on usage patterns
- Processing power limits maximum frame rate
- No obstacle detection or autonomous features

7. Applications

7.1 Practical Uses

- Remote surveillance and monitoring
- Hazardous environment exploration
- Educational demonstrations of IoT concepts
- Robotics research and development
- Home security applications
- Wildlife observation

7.2 Educational Value

- Understanding embedded systems programming
- Learning wireless communication protocols
- Practicing web development integration
- Exploring real-time data streaming
- Hardware-software integration experience

8. Future Enhancements

8.1 Proposed Improvements

Hardware Upgrades:

- Integration of ultrasonic sensors for obstacle avoidance
- Addition of servo motor for pan/tilt camera control
- Battery monitoring and low-power indicators
- GPS module for position tracking
- Accelerometer for tilt detection

Software Enhancements:

- Recording capability for video and commands
- Multiple concurrent video streams support
- Autonomous navigation algorithms
- Computer vision integration (object detection)
- Mobile application development
- Encryption for secure communication

Feature Additions:

- Night vision support using IR LEDs
- Two-way audio communication
- Route recording and playback
- Gesture-based control interface
- Voice command integration

8.2 Scalability Options

- Multi-robot coordination system
- Cloud integration for remote access
- Machine learning for path optimization
- Integration with home automation systems

9. Conclusion

This project successfully demonstrates the integration of multiple technologies to create a functional Wi-Fi controlled robot car with live video streaming capabilities. The ESP32 microcontroller proves to be an excellent platform for IoT applications, offering sufficient processing power, built-in Wi-Fi, and extensive GPIO capabilities.

The system achieves its primary objectives of remote control, real-time video transmission, and user-friendly operation through a web interface. The asynchronous architecture ensures responsive performance while the WebSocket protocol provides efficient bidirectional communication.

Key accomplishments include:

- Successful implementation of differential drive control
- Real-time video streaming with acceptable latency
- Intuitive touch-optimized web interface
- Robust WebSocket communication
- Efficient memory management using PSRAM

The project serves as an excellent foundation for further development in robotics, IoT, and embedded systems. Its modular design allows for easy expansion and integration of additional features.

10. References

Libraries and Frameworks

- ESP32 Arduino Core
- AsyncTCP Library
- ESPAsyncWebServer Library
- ESP Camera Library

Documentation

- ESP32 Technical Reference Manual
- WebSocket Protocol Specification (RFC 6455)
- HTML5 and CSS3 Standards
- Arduino Programming Reference

Hardware Specifications

- ESP32 Datasheet
- L298N Motor Driver Documentation
- ESP32-CAM Module Specifications

Appendix

A. Code Structure Overview

- **Setup Functions:** Pin configuration, Wi-Fi initialization, server setup
- **Motor Control:** Direction logic, PWM speed control
- **Camera Functions:** Initialization, frame capture, streaming
- **WebSocket Handlers:** Event processing, command parsing
- **Web Interface:** HTML structure, CSS styling, JavaScript logic

B. Troubleshooting Guide

- Camera initialization failures: Check pin connections and power supply

- Motor not responding: Verify motor driver connections and PWM channels
- Connection issues: Confirm Wi-Fi credentials and IP address
- Video lag: Reduce frame size or adjust JPEG quality
- Memory errors: Ensure PSRAM is properly initialized

C. Safety Considerations

- Ensure proper voltage regulation for components
- Implement current limiting for motor driver
- Add emergency stop functionality
- Use appropriate battery management
- Follow electrical safety protocols during assembly