Title: Automated Precision Task Using SWARM

Submitted By:

1.Gaurav Bhaltilak

2.Rohit Chouhan

3.Om Mangle

**1.Abstract**

The rapid evolution of the Internet of Things (IoT) has driven the demand for efficient, reliable, and low-latency communication systems that operate independently of traditional internet infrastructure. This project presents the design and implementation of a wireless communication system based on a Master-Slave architecture using ESP8266 NodeMCU microcontrollers. The communication between the master and slave devices is achieved using ESP-NOW, a proprietary connectionless communication protocol developed by Espressif, which allows multiple ESP8266 devices to exchange data directly without requiring a Wi-Fi router or internet connection.

The system is designed with two key objectives: remote actuation and real-time sensor data acquisition. The master ESP8266 functions as the central control unit that transmits movement commands—such as forward, backward, left, right, and stop—to the slave ESP8266. The slave, connected to a 4-wheel drive motor system via an L298N motor driver, executes these commands to control the motion of a robotic vehicle. Additionally, the slave is equipped with multiple sensors: a DHT11 sensor for measuring ambient temperature and humidity, and an HC-SR04 ultrasonic sensor for obstacle distance measurement. Upon executing each command, the slave collects real-time environmental data and transmits it back to the master node using ESP-NOW.

To further enhance the system’s functionality, the master device is integrated with Firebase Realtime Database, allowing it to store and remotely display sensor readings over the internet. This feature enables users to monitor environmental conditions and operational parameters through a mobile device or web browser from anywhere in the world. Firebase’s cloud integration adds scalability and facilitates future enhancements such as data logging, user interfaces, and alert systems.

This project successfully demonstrates a cost-effective, low-power, and router-free communication framework for real-time control and monitoring applications. The use of ESP-NOW ensures ultra-low latency and seamless device-to-device communication, making it ideal for robotics, automation, environmental monitoring, and smart agriculture. The system's modularity, ease of replication, and cloud integration potential open up avenues for further research and practical deployment in both academic and industrial domains.

**2.Introduction**

The convergence of embedded systems, wireless communication technologies, and the Internet of Things (IoT) has brought about a transformative shift in how real-time control and data acquisition systems are designed and implemented. In today's increasingly connected world, there is a growing demand for smart, autonomous systems that can function in both connected and disconnected environments. Whether in smart homes, automated farms, industrial automation, or mobile robotics, the ability to wirelessly monitor and control systems in real time is a critical capability that continues to evolve rapidly.

At the heart of this transformation lies the ESP8266 NodeMCU, a low-cost microcontroller with built-in Wi-Fi capabilities, which has become a popular platform for both hobbyists and professionals. The ESP8266 simplifies the development of connected devices by offering a single-chip solution that integrates a full TCP/IP stack, GPIOs, and enough processing power to handle both sensing and control logic. As a result, it has significantly lowered the barrier to entry for building scalable and affordable IoT applications. This democratization of technology has led to a surge in innovation across domains such as home automation, robotics, environmental sensing, agricultural monitoring, and industrial control systems.

While traditional IoT implementations rely heavily on cloud services and require continuous internet connectivity, there are many situations where offline operation is essential. For instance, in remote agricultural fields, disaster-hit areas, or military zones, internet infrastructure may be unavailable, unreliable, or even undesirable due to security concerns. In such scenarios, conventional Wi-Fi communication—dependent on routers or access points—becomes a limitation rather than a convenience.

This project addresses this challenge by developing a Master-Slave Wireless Communication System using two ESP8266 microcontrollers, configured to communicate using ESP-NOW, a low-latency, peer-to-peer protocol developed by Espressif. ESP-NOW allows direct communication between ESP devices using their MAC addresses, eliminating the need for a router, hotspot, or any intermediary network hardware. It functions somewhat like Bluetooth but with the longer range, higher throughput, and greater stability associated with Wi-Fi, making it an ideal protocol for device-to-device (D2D) communication in constrained or infrastructure-less environments.

In this system, one ESP8266 functions as the master, acting as the command center. It sends control instructions (such as "forward", "backward", "left", "right", or "stop") to a slave ESP8266, which is mounted on a 4-wheel robotic platform. The slave interprets the command and actuates a motor driver (L298N) to move the robot accordingly. Additionally, the slave device is equipped with environmental sensors—a DHT11 temperature and humidity sensor and an HC-SR04 ultrasonic distance sensor. After executing each command, the slave gathers sensor readings and sends the data back to the master in real-time.

The master ESP8266 not only interprets and logs this feedback data but also uploads it to Firebase Realtime Database, enabling cloud-based monitoring. This hybrid approach merges the benefits of offline device-level control and online data availability, offering the best of both worlds—resilience in disconnected environments and insightful analytics when internet access is restored or available.

In essence, this project demonstrates a practical, scalable, and efficient method for remote robot control and environmental monitoring without relying on traditional internet infrastructure. It emphasizes the use of modern communication protocols like ESP-NOW to enable reliable, low-power, and long-range communication between smart devices. The system’s architecture reflects key IoT principles—interoperability, responsiveness, and autonomy—making it highly suitable for educational purposes, field experimentation, and real-world deployments in agriculture, security, and automation.

**3. Objective of the Project**

The primary objective of this project is to design and implement a wireless robotic control system based on a master-slave architecture using ESP8266 NodeMCU microcontrollers. The system aims to enable real-time command transmission and feedback communication between the two devices using the ESP-NOW protocol.

More specifically, the objectives include:

* Establishing a wireless command link from a master ESP8266 to a slave ESP8266 to control motion of a 4-wheel drive robot via commands like forward, backward, left, right, and stop.
* Utilizing the L298N motor driver on the slave to control the robotic platform based on commands received.
* Integrating DHT11 (temperature and humidity sensor) and HC-SR04 ultrasonic sensor on the slave ESP8266 to gather environmental data.
* Ensuring that the slave transmits sensor readings back to the master in real-time using ESP-NOW.
* Implementing Firebase Realtime Database in the master ESP8266 to log, store, and display received data in the cloud for remote access.
* Creating a fully bi-directional communication system that operates effectively without the need for a router or internet, while optionally allowing cloud integration for monitoring.

This objective encompasses aspects of wireless communication, motor control, sensor interfacing, and cloud data management—offering a comprehensive IoT solution

**4. Overview of the System**

This project consists of two ESP8266 devices configured in a Master-Slave architecture:

* The Master ESP8266 acts as the brain of the system. It sends directional commands to the slave based on user inputs (via serial monitor or potential UI) and receives environmental data in return. It also connects to the internet via Wi-Fi and syncs sensor data to Firebase.
* The Slave ESP8266 performs dual roles: executing motor commands using an L298N motor driver and collecting sensor data (temperature, humidity, and obstacle distance). It operates without requiring any internet connection.

Key Components:

* ESP8266 NodeMCU (x2) – Microcontrollers with built-in Wi-Fi.
* L298N Motor Driver – Controls four DC motors.
* 4-Wheel Robotic Chassis – Enables movement based on commands.
* DHT11 Sensor – Measures ambient temperature and humidity.
* HC-SR04 Ultrasonic Sensor – Measures distance to nearby obstacles.
* Firebase – Cloud platform to display sensor data remotely.

The system is designed to function autonomously in disconnected environments while providing optional cloud monitoring for extended use cases.

**5. Working Principle**

The system operates on a two-way communication loop using ESP-NOW:

Phase 1: Control Transmission (Master → Slave)

* The user inputs a command via the serial monitor connected to the Master ESP8266 (e.g., 'f' for forward, 'l' for left).
* This command is encoded into a structured data packet and transmitted to the Slave ESP8266 using ESP-NOW.
* ESP-NOW transmits data directly using MAC addresses, bypassing the need for a Wi-Fi router.

Phase 2: Command Execution (Slave side)

* The Slave ESP8266 decodes the received command and activates the appropriate motor control logic using the L298N driver, which energizes the motors in the desired direction.
* The robot moves as instructed (forward, backward, left, right, or stops).

Phase 3: Sensor Reading and Data Feedback (Slave → Master)

* After executing the motion command, the slave reads sensor data from the DHT11 and HC-SR04 sensors.
* The sensor readings (temperature, humidity, and distance) are packaged and transmitted back to the Master ESP8266 via ESP-NOW.

Phase 4: Cloud Integration (Master → Firebase)

* The Master ESP8266 receives the sensor data and pushes it to the Firebase Realtime Database using HTTP over Wi-Fi.
* The data is stored and updated in real-time, allowing monitoring via mobile or desktop applications.

This process creates a real-time wireless feedback loop, enabling both control and monitoring from a central node.

**6. Motivation**

The motivation for this project arises from the growing demand for decentralized, offline-capable IoT systems in a world increasingly reliant on cloud and internet services. However, in many applications such as:

* Agriculture (farms without Wi-Fi)
* Disaster zones
* Military field operations
* Remote surveillance and patrolling
* Industrial environments with limited connectivity

…the reliance on continuous internet availability becomes a limitation.

To overcome this, ESP-NOW offers a perfect solution by enabling low-latency, device-to-device communication without routers or access points. This project demonstrates that effective wireless control and monitoring can be achieved even in disconnected environments, thus opening doors for reliable and flexible deployment in remote scenarios.

Furthermore, the integration of cloud-based Firebase enables data logging and remote monitoring when internet becomes available, thus marrying offline control with online visualization—a powerful hybrid approach that increases the adaptability of the system.

**7. Significance of the Project**

This project showcases several valuable innovations and contributions to the fields of IoT, embedded systems, and wireless control:

* Decentralized Communication: The use of ESP-NOW removes the dependency on routers or network infrastructure, significantly reducing cost and increasing deployment flexibility.
* Real-time Bidirectional Data Flow: Unlike typical unidirectional sensor networks, this project implements a fully bidirectional communication model, allowing both command and response cycles.
* Robust Motor Control Integration: By integrating a 4-wheel drive platform with a slave microcontroller, the project offers a strong foundation for mobile robotics.
* Cloud-Enabled Monitoring: The integration with Firebase Realtime Database allows remote users to track environmental data live, demonstrating how IoT solutions can bridge offline and online systems.
* Scalability: The system can be extended to support multiple slave nodes, additional sensors, or even AI-based decision making based on collected data.
* Educational and Practical Value: From a learning perspective, the project combines knowledge of wireless communication, motor control, sensor interfacing, and cloud computing in a single cohesive framework. From an industrial perspective, it offers a deployable prototype for smart agriculture, warehouse automation, or security patrol systems.

In summary, this system stands as a powerful prototype for autonomous, sensor-based robotic systems that are capable of offline operation and cloud-based data visualization, with clear applicability in both academic research and real-world deployments

**8. Literature Survey**

In the evolving landscape of the Internet of Things (IoT), seamless and efficient communication between devices is paramount. Numerous studies and technological advancements have been made to explore and optimize the communication protocols that underpin IoT systems. Each protocol comes with its own advantages, limitations, and specific use cases. This section reviews the key communication methodologies employed in previous research and highlights the gap that this project aims to bridge.

**8.1 Wi-Fi-Based Communication (HTTP/MQTT)**

Traditional IoT systems heavily rely on Wi-Fi as a medium of communication. Hypertext Transfer Protocol (HTTP) and Message Queuing Telemetry Transport (MQTT) are widely used application-layer protocols for transmitting data over Wi-Fi.

* HTTP is simple and widely supported, but its stateless nature and request-response structure introduce overhead and latency, making it less suitable for real-time applications.
* MQTT, on the other hand, is a lightweight, publish-subscribe protocol optimized for low-bandwidth, high-latency networks. It allows efficient message broadcasting, making it a common choice for sensor networks and telemetry applications.

However, both protocols require an internet connection and a central broker or server, typically hosted on the cloud. This reliance introduces vulnerabilities in situations where network infrastructure is unavailable, such as in remote, disaster-prone, or off-grid environments. Studies such as those by L. Tan and N. Wang (2010) in "Future Internet of Things: Architecture and Trends" emphasized that internet dependency can become a bottleneck in mission-critical systems.

**8.2 Bluetooth and BLE (Bluetooth Low Energy)**

Bluetooth and its power-efficient variant, Bluetooth Low Energy (BLE), have been widely adopted for short-range, low-power communication, especially in consumer electronics and wearable technologies. BLE allows for efficient sensor-to-gateway communication in personal area networks (PANs), with very low energy consumption.

Despite its efficiency in power-sensitive applications, BLE is constrained by:

* Limited range (~10–50 meters depending on conditions),
* Low data throughput, and
* Limited number of concurrent connections (typically 7 devices).

In robotic applications or remote sensing environments, these limitations restrict BLE's practicality. Prior implementations, such as indoor navigation systems or proximity-based applications, show success with BLE, but these are not scalable for outdoor, multi-device, or real-time control systems.

**8.3 ESP-NOW Protocol**

ESP-NOW, introduced by Espressif Systems, is a proprietary protocol built on top of the Wi-Fi physical layer. It allows direct peer-to-peer communication between ESP devices without the need for a Wi-Fi router or access point. Communication is initiated via MAC addresses, with the ability to send data to multiple peers simultaneously in broadcast or unicast modes.

The key benefits observed in prior implementations include:

* Ultra-low latency (~1–3 ms),
* Reduced power consumption, and
* Extended range compared to BLE.

Research articles, such as “Wireless Sensor Networks Based on ESP-NOW Protocol” (Rodriguez et al., 2021), demonstrate the use of ESP-NOW in agriculture and industrial settings where router-free communication enhances system reliability and mobility. Similarly, projects involving drone swarms and robotic fleets have adopted ESP-NOW to facilitate decentralized coordination.

**8.4 Firebase for Cloud-Based Monitoring**

Firebase Realtime Database, developed by Google, is increasingly used in modern IoT systems due to its:

* Real-time synchronization,
* Cross-platform support, and
* Ease of integration with mobile and web applications.

Firebase supports RESTful APIs, allowing embedded systems to push or retrieve data with minimal code. Prior work such as “Smart Farming Using IoT and Firebase” (Kumar et al., 2020) shows Firebase’s effectiveness in delivering cloud-based insights from remote devices.

However, many such systems still rely on constant internet availability, which can pose challenges in field applications. Moreover, using Firebase in real-time robotics scenarios is uncommon due to latency introduced by the cloud round-trip time.

**8.5 Research Gap and Contribution of This Project**

While previous studies have demonstrated the use of Wi-Fi, Bluetooth, and cloud-based platforms independently, few have combined ESP-NOW's offline capabilities with Firebase’s online monitoring to deliver a hybrid communication model. The gap lies in the ability to maintain robust, low-latency control between devices in offline scenarios, while still retaining the capacity to log and analyze data in the cloud when internet access is available.

This project addresses that gap by:

* Utilizing ESP-NOW for direct, router-independent control between master and slave ESP8266 modules,
* Implementing real-time motor actuation through ESP commands,
* Simultaneously capturing sensor data (temperature, humidity, distance), and
* Uploading data to Firebase for remote analytics and monitoring.

This dual-mode operation enhances the system's resilience, flexibility, and field-readiness, making it suitable for IoT-based robotics, mobile sensing platforms, and offline automation systems.

**9. System Analysis**

**9.1 Existing System**

In conventional wireless control and monitoring systems, communication between microcontrollers or embedded devices typically relies on Wi-Fi-based networking or Bluetooth connectivity. These systems, while widely used, come with several inherent limitations:

* Wi-Fi-Based Systems: These require devices to be connected to the same local network via a router or access point. In such systems, data is transmitted using high-level protocols like HTTP or MQTT, which introduce latency, require internet access, and may involve complex configurations such as IP addressing, NAT traversal, and port forwarding. Moreover, in remote or mobile environments, internet availability can be intermittent or completely absent, rendering Wi-Fi-based systems ineffective.
* Bluetooth-Based Systems: Although Bluetooth offers a relatively simple pairing process and low power consumption, its range is limited (typically less than 10 meters), and it supports a limited number of simultaneous connections. Bluetooth is also susceptible to signal interference in environments crowded with 2.4 GHz devices, such as industrial or academic laboratories. Furthermore, its data transfer rate is lower than Wi-Fi or ESP-NOW, making it unsuitable for fast-paced, real-time control applications like robotic motion control.
* Router Dependency: Many existing systems are heavily dependent on centralized infrastructure (e.g., routers, cloud brokers), which makes them less suitable for outdoor, field, or mobile deployments where such infrastructure may be unavailable.

These shortcomings create a barrier to scalability, robustness, and mobility, especially for applications like robotics, environmental sensing, or disaster management.

**9.2 Proposed System**

To overcome the above limitations, the proposed system utilizes a decentralized wireless communication framework based on ESP-NOW, a protocol developed by Espressif for their ESP8266/ESP32 series microcontrollers.

Key Features of the Proposed System:

* ESP-NOW Communication: The system enables direct peer-to-peer communication between a master ESP8266 (controller) and a slave ESP8266 (actuator + sensor node). The master sends control commands (e.g., forward, backward, left, right, stop) directly to the slave, which in turn actuates motors and responds with sensor data.
* Offline Operation: ESP-NOW does not require a Wi-Fi network or internet connection. It uses MAC address-based unicast or broadcast messaging over the Wi-Fi PHY layer, ensuring that the devices can communicate even in completely offline scenarios.
* Sensor Integration: The slave ESP8266 reads data from environmental sensors (DHT11 for temperature/humidity and ultrasonic sensor for distance measurement), making it suitable for monitoring and control applications.
* Firebase Integration: While real-time control occurs over ESP-NOW, the master device is connected to the internet and is capable of uploading the sensor data received from the slave to a Firebase Realtime Database. This creates a hybrid architecture—offline for control, online for data logging and analytics.
* User Interaction via Serial Interface: The user can input commands directly into the master ESP8266 using the serial terminal (e.g., pressing ‘f’ to move forward), and receive acknowledgment and sensor feedback almost instantly.

**9.3 Advantages of the Proposed System**

The architecture and technologies used in this system offer several distinct advantages over traditional implementations:

Low Latency Communication

* ESP-NOW operates over the lower-level MAC layer, enabling sub-millisecond transmission times.
* Commands from the master are received and executed by the slave almost instantly, making the system ideal for real-time robotics and automation.

Reduced Power Consumption

* ESP-NOW minimizes overhead by bypassing the TCP/IP stack, which leads to lower CPU usage and reduced energy consumption.
* This makes the system highly suitable for battery-operated or solar-powered deployments.

Simplified Network Setup

* The system eliminates the need for routers, SSIDs, and password authentication.
* Devices are pre-configured to recognize each other via MAC addresses, allowing for plug-and-play deployment with minimal setup time.

Robust in Offline or Remote Environments

* Unlike Wi-Fi or cloud-based systems, this architecture works in remote or disconnected environments such as forests, farms, or mobile robots on the move.
* This is critical for autonomous systems where connectivity is unpredictable.

Real-Time Data Monitoring via Firebase

* Although control is offline, sensor data can be forwarded to Firebase through the master node, allowing:
  + Remote monitoring
  + Data logging
  + Historical analysis
  + Mobile or web dashboard integration

Scalability and Expandability

* The system can be scaled to control multiple slave devices by assigning different MAC addresses to each node.
* It is easily expandable to integrate more sensors or actuators for complex multi-agent systems.

Cost-Effective and Open-Source

* All hardware used (ESP8266, DHT11, Ultrasonic sensor, L298N motor driver) is low-cost and widely available.
* The software stack (Arduino, Firebase, ESP-NOW) is completely open-source, reducing licensing issues.

**10. Implementation**

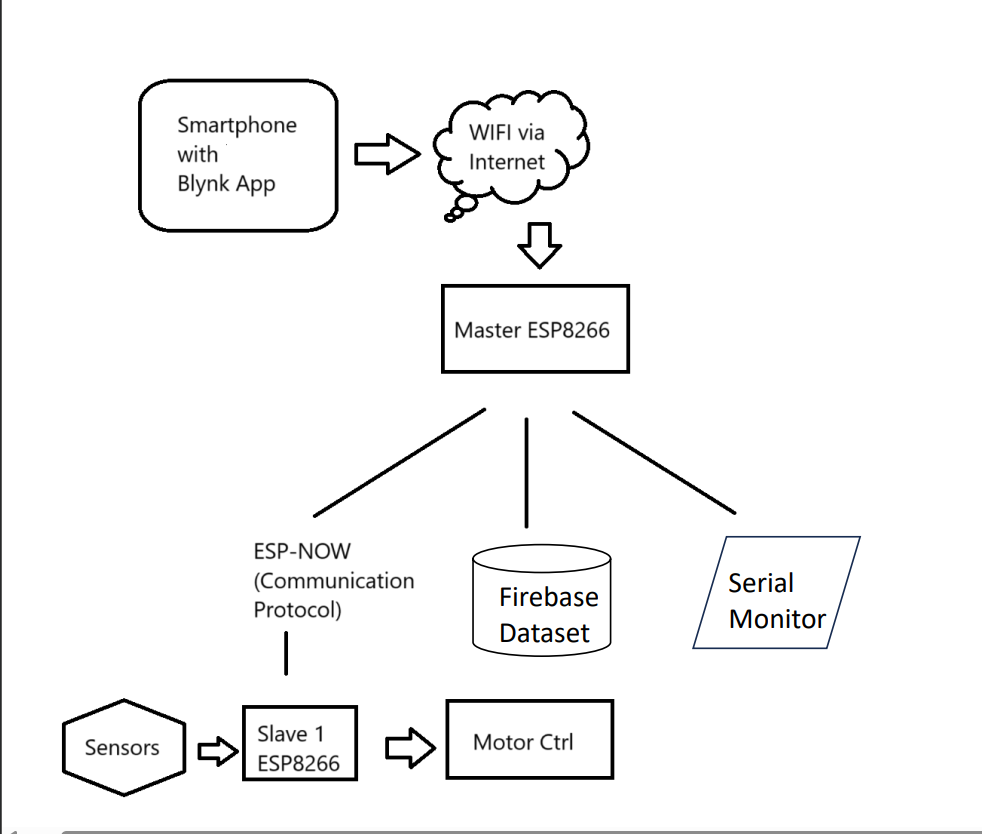
Implementation

The project is implemented using two ESP8266 modules (NodeMCU) communicating via ESP-NOW protocol. The master module is connected to the Blynk app and is responsible for sending control signals. The slave module receives these signals and uses them to operate motors and gather environmental data from the DHT11 (temperature and humidity sensor) and an ultrasonic distance sensor.

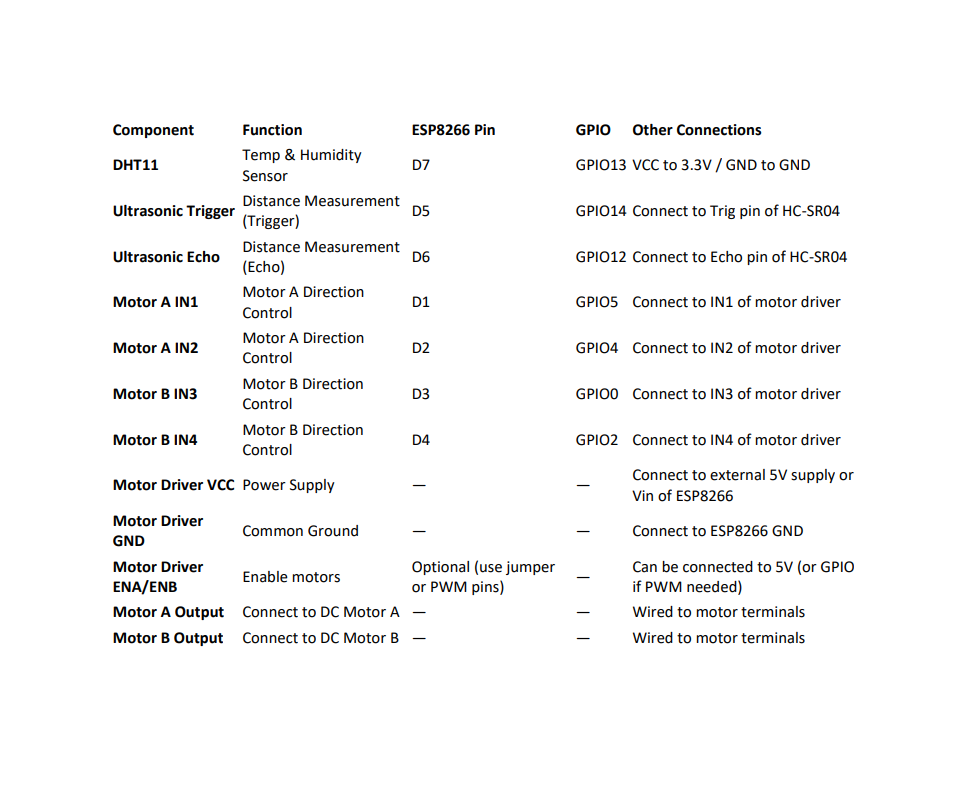
Steps:

1. **Hardware Setup:**
   * **ESP8266 master connected via Wi-Fi to Blynk app.**
   * ESP8266 slave connected to DHT11, ultrasonic sensor, and motor driver (L298N).
   * Motors controlled based on commands from the master.
2. Communication:
   * ESP-NOW protocol is configured for wireless communication between master and slave.
3. Programming:
   * Master: Reads user inputs from Blynk and sends control messages.
   * Slave: Receives commands, operates motors accordingly, and reads data from sensors.
4. Feedback (Optional):
   * Sensor data can be sent back to the master for display in the Blynk app.

**10.1 Block Diagram**

****

**10.2 Connections**

****

**10.3 Code-**

**Master code –**

**#define BLYNK\_TEMPLATE\_ID "TMPL3NxcnEp8E"**

**#define BLYNK\_TEMPLATE\_NAME "Master"**

**#define BLYNK\_AUTH\_TOKEN "IWM1WMVgD7xPqmL3e5oR1jTDRGognJCx"**

**#include <ESP8266WiFi.h>**

**#include <espnow.h>**

**#include <FirebaseESP8266.h>**

**#include <BlynkSimpleEsp8266.h>**

**// --- WiFi Credentials ---**

**const char\* ssid = "motoedge50fusion\_6126";**

**const char\* password = "4qmdtnym";**

**// --- Firebase Credentials ---**

**#define FIREBASE\_HOST "esp8266-ac2e8-default-rtdb.firebaseio.com"**

**#define FIREBASE\_AUTH "e3zp8esTYo6yGdhxY3GeLrvAZwCo08Fl3PgmhpzG"**

**// --- Blynk Auth Token ---**

**char auth[] = "IWM1WMVgD7xPqmL3e5oR1jTDRGognJCx"; // Replace with your Blynk token**

**// --- Slave MAC Address ---**

**uint8\_t slaveAddress[] = {0x4C, 0xEB, 0xD6, 0x1F, 0x86, 0x8A};**

**// --- ESP-NOW Structs ---**

**typedef struct struct\_command {**

**int commandCode;**

**} struct\_command;**

**typedef struct struct\_sensorData {**

**float temperature;**

**float humidity;**

**float distance;**

**} struct\_sensorData;**

**struct\_command commandToSend;**

**struct\_sensorData receivedSensorData;**

**bool dataReceived = false;**

**// --- Firebase Objects ---**

**FirebaseData firebaseData;**

**FirebaseAuth authFb;**

**FirebaseConfig config;**

**// --- ESP-NOW Send Callback ---**

**void OnDataSent(uint8\_t \*mac\_addr, uint8\_t sendStatus) {**

**Serial.print("Command Delivery Status: ");**

**Serial.println(sendStatus == 0 ? "✅ Success" : "❌ Fail");**

**}**

**// --- ESP-NOW Receive Callback ---**

**void OnDataRecv(uint8\_t \*mac, uint8\_t \*incomingData, uint8\_t len) {**

**if (len == sizeof(struct\_sensorData)) {**

**memcpy(&receivedSensorData, incomingData, sizeof(receivedSensorData));**

**dataReceived = true;**

**} else {**

**Serial.println("❗ Unexpected data size!");**

**}**

**}**

**// --- Send Command Helper ---**

**void sendCommand(int code) {**

**commandToSend.commandCode = code;**

**esp\_now\_send(slaveAddress, (uint8\_t \*)&commandToSend, sizeof(commandToSend));**

**}**

**// --- Blynk Virtual Button Handlers ---**

**BLYNK\_WRITE(V0) { if (param.asInt()) { sendCommand(1); Serial.println("➡ Forward (Blynk)"); } }**

**BLYNK\_WRITE(V1) { if (param.asInt()) { sendCommand(2); Serial.println("⬅ Backward (Blynk)"); } }**

**BLYNK\_WRITE(V2) { if (param.asInt()) { sendCommand(3); Serial.println("↩ Left (Blynk)"); } }**

**BLYNK\_WRITE(V3) { if (param.asInt()) { sendCommand(4); Serial.println("↪ Right (Blynk)"); } }**

**BLYNK\_WRITE(V4) { if (param.asInt()) { sendCommand(0); Serial.println("⏹ Stop (Blynk)"); } }**

**void setup() {**

**Serial.begin(115200);**

**delay(100);**

**WiFi.begin(ssid, password);**

**Serial.print("🔌 Connecting to Wi-Fi");**

**while (WiFi.status() != WL\_CONNECTED) {**

**delay(500); Serial.print(".");**

**}**

**Serial.println("\n✅ Wi-Fi Connected");**

**Serial.print("🌐 IP: "); Serial.println(WiFi.localIP());**

**// Init Firebase**

**config.database\_url = FIREBASE\_HOST;**

**config.signer.tokens.legacy\_token = FIREBASE\_AUTH;**

**Firebase.begin(&config, &authFb);**

**Firebase.reconnectWiFi(true);**

**// Init Blynk**

**Blynk.begin(auth, ssid, password);**

**// Init ESP-NOW**

**if (esp\_now\_init() != 0) {**

**Serial.println("❌ ESP-NOW init failed");**

**return;**

**}**

**esp\_now\_set\_self\_role(ESP\_NOW\_ROLE\_CONTROLLER);**

**esp\_now\_register\_send\_cb(OnDataSent);**

**esp\_now\_register\_recv\_cb(OnDataRecv);**

**esp\_now\_add\_peer(slaveAddress, ESP\_NOW\_ROLE\_SLAVE, 1, NULL, 0);**

**Serial.println("✅ Master Ready (Blynk + Firebase + ESP-NOW)");**

**}**

**void loop() {**

**Blynk.run();**

**// Handle Serial Input (optional)**

**if (Serial.available()) {**

**char input = Serial.read();**

**switch (input) {**

**case 'f': sendCommand(1); Serial.println("➡ Forward"); break;**

**case 'b': sendCommand(2); Serial.println("⬅ Backward"); break;**

**case 'l': sendCommand(3); Serial.println("↩ Left"); break;**

**case 'r': sendCommand(4); Serial.println("↪ Right"); break;**

**case 's': sendCommand(0); Serial.println("⏹ Stop"); break;**

**default: Serial.println("❗ Invalid input (f/b/l/r/s)"); return;**

**}**

**}**

**// Upload sensor data to Firebase**

**if (dataReceived) {**

**dataReceived = false;**

**Serial.println("\n📩 Sensor Data Received:");**

**Serial.print("🌡 Temperature: "); Serial.println(receivedSensorData.temperature);**

**Serial.print("💧 Humidity: "); Serial.println(receivedSensorData.humidity);**

**Serial.print("📏 Distance: "); Serial.println(receivedSensorData.distance);**

**delay(200); // For stability**

**if (Firebase.ready()) {**

**FirebaseJson json;**

**json.add("temperature", receivedSensorData.temperature);**

**json.add("humidity", receivedSensorData.humidity);**

**json.add("distance", receivedSensorData.distance);**

**json.add("timestamp", (int)time(nullptr));**

**if (Firebase.pushJSON(firebaseData, "/sensors/data", json)) {**

**Serial.println("✅ Data uploaded to Firebase");**

**} else {**

**Serial.println("❌ Firebase upload failed: " + firebaseData.errorReason());**

**}**

**}**

**}**

**}**

Slave Code-

#include <ESP8266WiFi.h>

#include <espnow.h>

#include <DHT.h>

// ----- DHT11 Setup -----

#define DHTPIN D7 // GPIO13

#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

// ----- Ultrasonic Sensor -----

#define TRIG\_PIN D5 // GPIO14

#define ECHO\_PIN D6 // GPIO12

// ----- Motor Pins -----

#define IN1 D1

#define IN2 D2

#define IN3 D3

#define IN4 D4

// ----- Structures for ESP-NOW -----

typedef struct struct\_command {

int commandCode;

} struct\_command;

typedef struct struct\_sensorData {

float temperature;

float humidity;

float distance;

} struct\_sensorData;

struct\_command receivedCommand;

struct\_sensorData sensorDataToSend;

// ----- Flags and Variables -----

volatile bool newCommandReceived = false;

uint8\_t masterMac[6];

unsigned long lastSensorSendTime = 0;

const unsigned long sensorInterval = 10000;

unsigned long commandStartTime = 0;

bool commandActive = false;

const unsigned long motorRunTime = 3000;

int currentCommand = 0;

// Precise turn duration (adjust experimentally)

const unsigned long TURN\_DURATION\_MS = 300; // ~90-degree turn timing

// ----- Get Distance -----

float getDistanceCM() {

digitalWrite(TRIG\_PIN, LOW);

delayMicroseconds(2);

digitalWrite(TRIG\_PIN, HIGH);

delayMicroseconds(10);

digitalWrite(TRIG\_PIN, LOW);

long duration = pulseIn(ECHO\_PIN, HIGH);

float distance = duration \* 0.034 / 2;

return distance;

}

// ----- Motor Control -----

void executeCommand(int command) {

switch (command) {

case 1: // Forward

digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);

digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);

break;

case 2: // Backward

digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);

digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);

break;

case 3: // Left (Precise 90° Turn)

digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);

digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);

delay(TURN\_DURATION\_MS);

executeCommand(0); // Stop after turning

commandActive = false;

currentCommand = 0;

Serial.println("↩ Left Turn Completed");

break;

case 4: // Right (Precise 90° Turn)

digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);

digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);

delay(TURN\_DURATION\_MS);

executeCommand(0); // Stop after turning

commandActive = false;

currentCommand = 0;

Serial.println("↪ Right Turn Completed");

break;

default: // Stop

digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);

digitalWrite(IN3, LOW); digitalWrite(IN4, LOW);

break;

}

}

// ----- ESP-NOW Receive Callback -----

void OnDataRecv(uint8\_t \*mac, uint8\_t \*incomingData, uint8\_t len) {

if (len == sizeof(receivedCommand)) {

memcpy(&receivedCommand, incomingData, sizeof(receivedCommand));

memcpy(masterMac, mac, 6);

currentCommand = receivedCommand.commandCode;

commandStartTime = millis();

commandActive = true;

newCommandReceived = true;

Serial.print("📥 Command Received: ");

Serial.println(currentCommand);

} else {

Serial.println("⚠ Invalid data size received!");

}

}

// ----- Setup -----

void setup() {

Serial.begin(115200);

pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);

pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);

pinMode(TRIG\_PIN, OUTPUT); pinMode(ECHO\_PIN, INPUT);

dht.begin();

WiFi.mode(WIFI\_STA);

WiFi.begin("motoedge50fusion\_6126", "4qmdtnym");

Serial.print("🔌 Connecting to Wi-Fi");

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("\n✅ Connected to Wi-Fi");

int channel = WiFi.channel();

Serial.print("📡 Wi-Fi Channel: ");

Serial.println(channel);

WiFi.disconnect();

WiFi.mode(WIFI\_STA);

wifi\_set\_channel(channel);

if (esp\_now\_init() != 0) {

Serial.println("❌ ESP-NOW initialization failed!");

return;

}

Serial.println("✅ ESP-NOW initialized");

esp\_now\_set\_self\_role(ESP\_NOW\_ROLE\_SLAVE);

esp\_now\_register\_recv\_cb(OnDataRecv);

Serial.println("✅ ESP-NOW Slave Ready");

}

// ----- Loop -----

void loop() {

// Stop motors after timeout (only for forward/backward)

if (commandActive && millis() - commandStartTime >= motorRunTime &&

(currentCommand == 1 || currentCommand == 2)) {

executeCommand(0);

Serial.println("⏹ Forward/Backward Command Timed Out. Motors stopped.");

commandActive = false;

currentCommand = 0;

}

// Handle new command

if (newCommandReceived) {

newCommandReceived = false;

executeCommand(currentCommand);

// Send sensor data if interval met

unsigned long currentTime = millis();

if (currentTime - lastSensorSendTime >= sensorInterval) {

lastSensorSendTime = currentTime;

float temp = dht.readTemperature();

float hum = dht.readHumidity();

float dist = getDistanceCM();

if (isnan(temp) || isnan(hum)) {

Serial.println("⚠ Failed to read from DHT sensor!");

return;

}

sensorDataToSend.temperature = temp;

sensorDataToSend.humidity = hum;

sensorDataToSend.distance = dist;

esp\_now\_send(masterMac, (uint8\_t \*)&sensorDataToSend, sizeof(sensorDataToSend));

Serial.println("📤 Sensor Data Sent:");

Serial.print(" 🌡 Temp: "); Serial.println(temp);

Serial.print(" 💧 Humidity: "); Serial.println(hum);

Serial.print(" 📏 Distance: "); Serial.println(dist);

} else {

Serial.println("⏳ Skipping sensor send (interval not met)");

}

}

}

**11. Result**

* The system successfully enabled remote control of motors using the Blynk mobile application.
* Sensor readings from the DHT11 and ultrasonic sensor were accurately captured on the slave device.
* ESP-NOW provided reliable communication between the two ESP8266 boards without the need for a Wi-Fi router.
* The project demonstrated real-time monitoring and control functionality using a low-cost, energy-efficient system.

**12. Future Scope/Enhancements**

1. **Two-Way Communication:**
   * **Implement feedback from the slave to the master to display sensor data in real-time on the Blynk app.**
2. **Obstacle Avoidance:**
   * **Integrate the ultrasonic sensor readings into logic that prevents collisions.**
3. Battery Monitoring:
   * Add voltage sensors to track power levels and send alerts via Blynk.
4. Expand to IoT Cloud:
   * Enable cloud data logging and analytics using platforms like Thingspeak or Firebase.
5. Voice Control Integration:
   * Use Google Assistant or Alexa for hands-free control.
6. Add More Sensors:
   * Incorporate gas, flame, or motion sensors for more complex applications.

**12.Conclusion**

The project successfully demonstrated wireless communication between two ESP8266 devices using ESP-NOW. The master module, interfaced with the Blynk app, controlled the motors, while the slave handled sensor data acquisition and actuator control. This implementation is low-cost, scalable, and suitable for various IoT applications such as robotics, smart homes, and environmental monitoring.