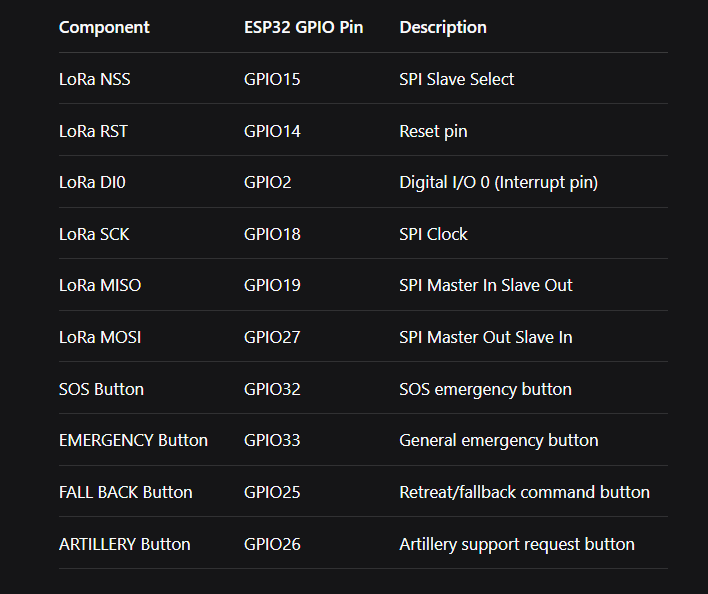
**Objective**

Create a military-grade long-range communication system using ESP32 with LoRa (RA-02 module) and four push buttons for emergency communications. The system should:

* Establish a WiFi access point for device monitoring
* Enable long-range LoRa communication (2-3 km line of sight)
* Provide four physical buttons for emergency messages (SOS, EMERGENCY, FALL BACK, ARTILLERY)
* Offer a web interface for status monitoring
* Ensure reliable communication in field conditions.

**Pin Configuration**

****

**Code**

#include <WiFi.h>

#include <WebServer.h>

#include <SPI.h>

#include <LoRa.h>

// Define button pins

const int buttonPins[] = {12, 13, 25, 26};

const int numButtons = 4;

// LoRa pins

#define LORA\_RST 14 // D14

#define LORA\_DIO0 2 // D2

#define LORA\_NSS 15 // D15

#define LORA\_MOSI 27 // D27

#define LORA\_MISO 19 // D19

#define LORA\_SCK 18 // D18

// Access Point credentials

const char\* ap\_ssid = "LoRaButtonMonitor";

const char\* ap\_password = "lora12345";

// Button states

int buttonStates[] = {0, 0, 0, 0};

int lastButtonStates[] = {0, 0, 0, 0};

bool buttonPressed[] = {false, false, false, false};

// Messages for each button

const char\* messages[] = {

"EMERGENCY STOP ACTIVATED",

"FALLBACK PROTOCOL INITIATED",

"SYSTEM WARNING: CHECK STATUS",

"SAFETY MODE ENGAGED"

};

// Web server

WebServer server(80);

// LoRa variables

String loraMessage = "";

bool newLoraMessage = false;

// Device ID for LoRa network

String deviceId = "";

void setup() {

Serial.begin(115200);

// Generate a unique device ID

deviceId = String((uint32\_t)ESP.getEfuseMac(), HEX);

Serial.println("Device ID: " + deviceId);

// Initialize button pins as inputs with pull-up resistors

for (int i = 0; i < numButtons; i++) {

pinMode(buttonPins[i], INPUT\_PULLUP);

}

// Initialize LoRa

initializeLoRa();

// Set up Access Point

WiFi.softAP(ap\_ssid, ap\_password);

IPAddress myIP = WiFi.softAPIP();

Serial.print("AP IP address: ");

Serial.println(myIP);

// Set up server routes

server.on("/", handleRoot);

server.on("/status", handleStatus);

server.on("/lora", handleLora);

server.on("/config", handleConfig);

// Start server

server.begin();

Serial.println("HTTP server started");

}

void initializeLoRa() {

// Initialize LoRa module

SPI.begin(LORA\_SCK, LORA\_MISO, LORA\_MOSI, LORA\_NSS);

LoRa.setPins(LORA\_NSS, LORA\_RST, LORA\_DIO0);

if (!LoRa.begin(433E6)) { // Initialize at 433 MHz

Serial.println("LoRa init failed. Check your connections.");

while (true);

}

LoRa.setSyncWord(0xF3); // Set sync word for private network

LoRa.setSpreadingFactor(7); // Lower spreading factor for faster transmission

LoRa.onReceive(onLoraReceive);

LoRa.receive(); // Put LoRa in receive mode

Serial.println("LoRa initialized successfully!");

}

void onLoraReceive(int packetSize) {

if (packetSize == 0) return;

// Read packet

loraMessage = "";

while (LoRa.available()) {

loraMessage += (char)LoRa.read();

}

newLoraMessage = true;

Serial.print("LoRa Received: ");

Serial.println(loraMessage);

}

void loop() {

server.handleClient();

// Check each button

for (int i = 0; i < numButtons; i++) {

// Read the button state

buttonStates[i] = digitalRead(buttonPins[i]);

// Check if button state changed from high to low (pressed)

if (buttonStates[i] == LOW && lastButtonStates[i] == HIGH) {

Serial.print("Button ");

Serial.print(i);

Serial.print(" pressed: ");

Serial.println(messages[i]);

// Send message via LoRa

String loraMsg = "BTN\_" + String(i) + ":" + messages[i] + ":" + deviceId;

sendLoRaMessage(loraMsg);

buttonPressed[i] = true;

// Small delay to debounce

delay(50);

}

// Update last button state

lastButtonStates[i] = buttonStates[i];

}

// Process received LoRa messages

if (newLoraMessage) {

processLoraMessage(loraMessage);

newLoraMessage = false;

}

// Small delay to prevent flooding

delay(10);

}

void sendLoRaMessage(String message) {

LoRa.beginPacket();

LoRa.print(message);

LoRa.endPacket();

Serial.println("Sent via LoRa: " + message);

// Return to receive mode

LoRa.receive();

}

void processLoraMessage(String message) {

// Parse received message

if (message.startsWith("BTN\_")) {

int firstSeparator = message.indexOf(':');

int secondSeparator = message.indexOf(':', firstSeparator + 1);

if (firstSeparator != -1 && secondSeparator != -1) {

String btnNumStr = message.substring(4, firstSeparator);

String btnMsg = message.substring(firstSeparator + 1, secondSeparator);

String senderId = message.substring(secondSeparator + 1);

int btnNum = btnNumStr.toInt();

if (btnNum >= 0 && btnNum < numButtons) {

buttonPressed[btnNum] = true;

Serial.println("LoRa button press from " + senderId + ": " + btnMsg);

}

}

}

}

void handleRoot() {

String html = R"=====(

<!DOCTYPE html>

<html lang='en'>

<head>

<meta charset='UTF-8'>

<meta name='viewport' content='width=device-width, initial-scale=1.0'>

<title>ESP32 LoRa Button Monitor</title>

<style>

\* { margin: 0; padding: 0; box-sizing: border-box; font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif; }

body { background: linear-gradient(135deg, #1a2a6c, #b21f1f, #fdbb2d); min-height: 100vh; display: flex; flex-direction: column; align-items: center; padding: 20px; color: white; }

.container { width: 100%; max-width: 900px; background: rgba(0, 0, 0, 0.7); border-radius: 15px; padding: 20px; box-shadow: 0 10px 30px rgba(0, 0, 0, 0.5); }

header { text-align: center; margin-bottom: 30px; padding-bottom: 15px; border-bottom: 2px solid rgba(255, 255, 255, 0.2); }

h1 { font-size: 2.5rem; margin-bottom: 10px; text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5); }

.status { display: flex; justify-content: space-between; margin-bottom: 20px; }

.connection-status { display: flex; align-items: center; font-weight: bold; }

.status-indicator { width: 15px; height: 15px; border-radius: 50%; margin-right: 10px; }

.connected { background-color: #2ecc71; box-shadow: 0 0 10px #2ecc71; }

.buttons-grid { display: grid; grid-template-columns: repeat(auto-fit, minmax(200px, 1fr)); gap: 20px; margin-bottom: 30px; }

.button-card { background: rgba(255, 255, 255, 0.1); border-radius: 10px; padding: 15px; text-align: center; transition: all 0.3s ease; }

.button-card.active { background: rgba(255, 255, 255, 0.2); transform: scale(1.05); box-shadow: 0 0 20px rgba(255, 255, 255, 0.3); }

.button-label { font-size: 1.2rem; margin-bottom: 10px; color: #ffcc00; }

.button-desc { font-size: 0.9rem; opacity: 0.8; }

.messages { background: rgba(0, 0, 0, 0.5); border-radius: 10px; padding: 20px; height: 300px; overflow-y: auto; }

.messages-header { display: flex; justify-content: space-between; margin-bottom: 15px; padding-bottom: 10px; border-bottom: 1px solid rgba(255, 255, 255, 0.2); }

.message-list { list-style: none; }

.message { padding: 10px; margin-bottom: 10px; border-radius: 5px; background: rgba(255, 255, 255, 0.1); animation: fadeIn 0.5s ease; }

.message.emergency { background: rgba(231, 76, 60, 0.3); border-left: 5px solid #e74c3c; }

.message.fallback { background: rgba(241, 196, 15, 0.3); border-left: 5px solid #f1c40f; }

.message.warning { background: rgba(52, 152, 219, 0.3); border-left: 5px solid #3498db; }

.message.safety { background: rgba(46, 204, 113, 0.3); border-left: 5px solid #2ecc71; }

.timestamp { font-size: 0.8rem; opacity: 0.7; }

.message-text { margin-top: 5px; font-weight: bold; }

.lora-status { background: rgba(255, 255, 255, 0.1); padding: 10px; border-radius: 5px; margin-top: 15px; }

.device-info { background: rgba(255, 255, 255, 0.1); padding: 10px; border-radius: 5px; margin-top: 15px; }

@keyframes fadeIn { from { opacity: 0; transform: translateY(10px); } to { opacity: 1; transform: translateY(0); } }

.clear-btn { background: #e74c3c; color: white; border: none; padding: 8px 15px; border-radius: 5px; cursor: pointer; transition: background 0.3s; }

.clear-btn:hover { background: #c0392b; }

footer { margin-top: 20px; text-align: center; font-size: 0.9rem; opacity: 0.7; }

</style>

</head>

<body>

<div class='container'>

<header>

<h1>ESP32 LoRa Button Monitor</h1>

<p>Connect to WiFi: <strong>LoRaButtonMonitor</strong> | Password: <strong>lora12345</strong></p>

</header>

<div class='status'>

<div class='connection-status'>

<div class='status-indicator connected'></div>

<span>Connected to LoRa Network</span>

</div>

<div>Messages: <span id='message-count'>0</span></div>

</div>

<div class='device-info'>

<strong>Device ID:</strong> <span id='device-id'>Loading...</span>

</div>

<div class='buttons-grid'>

<div class='button-card' id='button-0'>

<div class='button-label'>Button 1</div>

<div class='button-desc'>EMERGENCY STOP</div>

</div>

<div class='button-card' id='button-1'>

<div class='button-label'>Button 2</div>

<div class='button-desc'>FALLBACK PROTOCOL</div>

</div>

<div class='button-card' id='button-2'>

<div class='button-label'>Button 3</div>

<div class='button-desc'>SYSTEM WARNING</div>

</div>

<div class='button-card' id='button-3'>

<div class='button-label'>Button 4</div>

<div class='button-desc'>SAFETY MODE</div>

</div>

</div>

<div class='lora-status'>

<strong>LoRa Status:</strong> Active (433 MHz) | <strong>Sync Word:</strong> 0xF3

</div>

<div class='messages'>

<div class='messages-header'>

<h2>Message Log</h2>

<button class='clear-btn' onclick='clearLog()'>Clear Log</button>

</div>

<ul class='message-list' id='message-list'>

<li class='message'>

<div class='timestamp'>System: </div>

<div class='message-text'>Waiting for button signals...</div>

</li>

</ul>

</div>

</div>

<footer>

<p>ESP32 LoRa Button Monitor System | All devices connected via LoRa</p>

</footer>

<script>

// DOM elements

const messageList = document.getElementById('message-list');

const messageCount = document.getElementById('message-count');

const deviceIdElement = document.getElementById('device-id');

const buttons = [

document.getElementById('button-0'),

document.getElementById('button-1'),

document.getElementById('button-2'),

document.getElementById('button-3')

];

// Message classes for styling

const messageClasses = ['emergency', 'fallback', 'warning', 'safety'];

// Device ID

let deviceId = '';

// Add message to log

function addMessage(text, buttonId, sourceDevice, isLocal = false) {

const now = new Date();

const timestamp = now.toLocaleTimeString();

const li = document.createElement('li');

li.className = 'message ' + messageClasses[buttonId];

let sourceText = sourceDevice;

if (isLocal) {

sourceText += ' (You)';

}

li.innerHTML = `

<div class="timestamp">${timestamp} (${sourceText}):</div>

<div class="message-text">${text}</div>

`;

messageList.prepend(li);

updateMessageCount();

}

// Visual feedback for button press

function visualFeedback(buttonId) {

buttons[buttonId].classList.add('active');

setTimeout(() => {

buttons[buttonId].classList.remove('active');

}, 500);

}

// Update message count

function updateMessageCount() {

const count = messageList.querySelectorAll('li').length;

messageCount.textContent = count;

}

// Clear message log

function clearLog() {

messageList.innerHTML = '';

updateMessageCount();

addSystemMessage('Log cleared');

}

// Add system message

function addSystemMessage(text) {

const now = new Date();

const timestamp = now.toLocaleTimeString();

const li = document.createElement('li');

li.className = 'message';

li.innerHTML = `

<div class="timestamp">${timestamp} (System):</div>

<div class="message-text">${text}</div>

`;

messageList.prepend(li);

}

// Check for button presses by polling the server

function checkButtonStatus() {

fetch('/status')

.then(response => response.json())

.then(data => {

// Update device ID

if (data.deviceId && deviceId === '') {

deviceId = data.deviceId;

deviceIdElement.textContent = deviceId;

}

for (let i = 0; i < data.buttons.length; i++) {

if (data.buttons[i].pressed) {

visualFeedback(i);

addMessage(data.buttons[i].message, i, deviceId, true);

}

}

// Check for LoRa messages

if (data.loraMessage) {

processLoraMessage(data.loraMessage);

}

})

.catch(error => console.error('Error:', error));

}

// Process LoRa message

function processLoraMessage(message) {

if (message.startsWith("BTN\_")) {

const parts = message.split(':');

if (parts.length >= 3) {

const btnNum = parseInt(parts[0].replace('BTN\_', ''));

const btnMsg = parts[1];

const senderId = parts[2];

if (btnNum >= 0 && btnNum < 4) {

visualFeedback(btnNum);

addMessage(btnMsg, btnNum, senderId, senderId === deviceId);

}

}

}

}

// Initialize when page loads

window.onload = function() {

// Check button status every 500ms

setInterval(checkButtonStatus, 500);

// Clear initial system message

setTimeout(() => {

if (messageList.children.length === 1) {

messageList.innerHTML = '';

updateMessageCount();

}

}, 2000);

};

</script>

</body>

</html>

)=====";

server.send(200, "text/html", html);

}

void handleStatus() {

String json = "{";

json += "\"deviceId\":\"";

json += deviceId;

json += "\",\"buttons\":[";

for (int i = 0; i < numButtons; i++) {

json += "{";

json += "\"pressed\":";

json += buttonPressed[i] ? "true" : "false";

json += ",\"message\":\"";

json += messages[i];

json += "\"}";

if (i < numButtons - 1) json += ",";

// Reset button pressed state after reporting

buttonPressed[i] = false;

}

json += "]";

// Add LoRa message if available

if (newLoraMessage) {

json += ",\"loraMessage\":\"";

json += loraMessage;

json += "\"";

newLoraMessage = false;

}

json += "}";

server.send(200, "application/json", json);

}

void handleLora() {

if (server.hasArg("message")) {

String message = server.arg("message");

sendLoRaMessage(message);

server.send(200, "text/plain", "Message sent via LoRa: " + message);

} else {

server.send(400, "text/plain", "Missing message parameter");

}

}

void handleConfig() {

String json = "{";

json += "\"ssid\":\"";

json += ap\_ssid;

json += "\",\"password\":\"";

json += ap\_password;

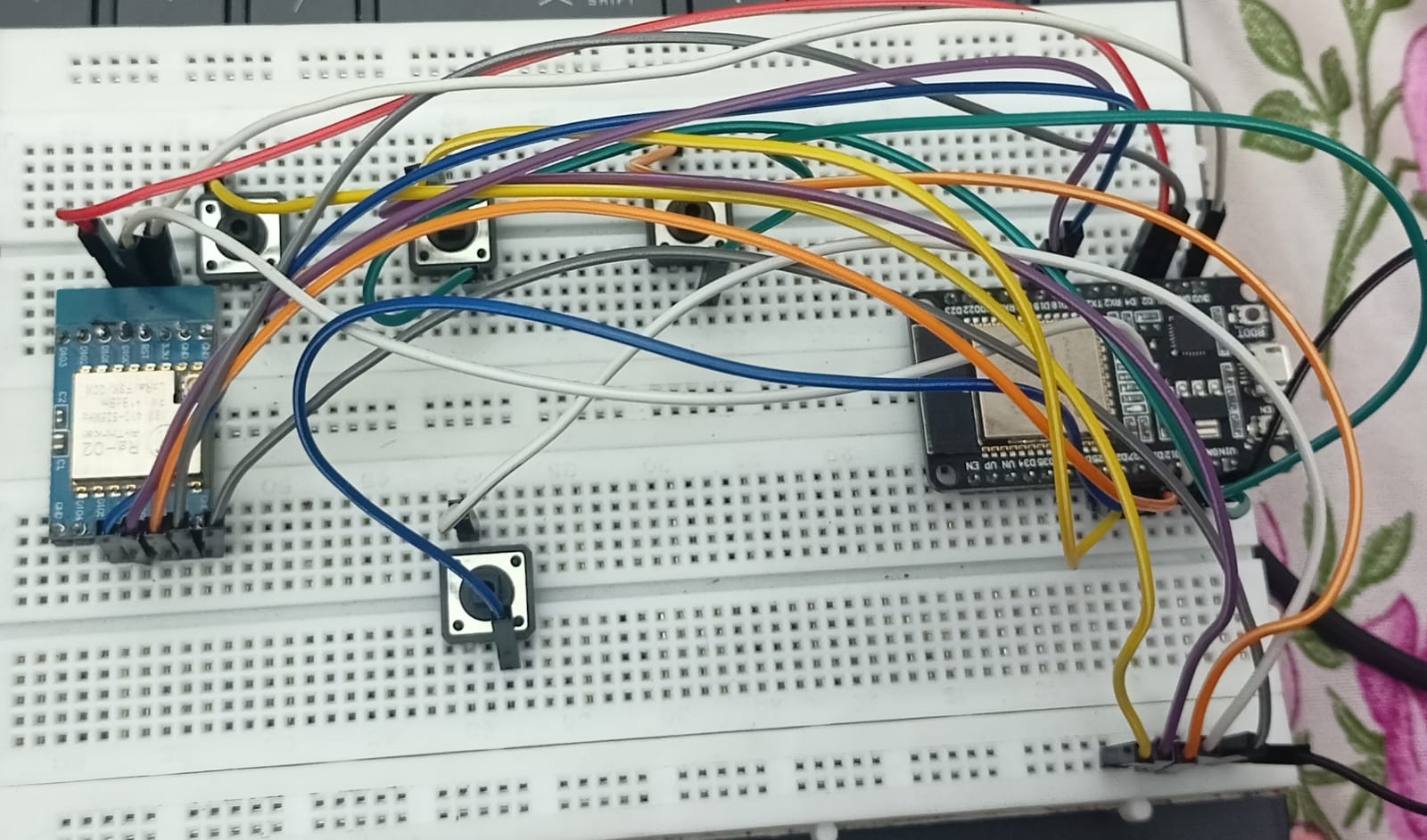
json += "\",\"deviceId\":\"";

json += deviceId;

json += "\"}";

server.send(200, "application/json", json);

}  
  
  
**Photos**



**Objective**

The primary objective of addressing language barriers in military operations is to ensure effective communication, coordination, and operational success across multinational forces, local populations, and allied partners in diverse linguistic environments.

* Enable real-time translation for soldiers in their chosen language.
* Ensure clear communication during emergencies.
* Reduce language barriers to improve coordination.
* Train an AI model to accurately translate voice commands.

**Code**

**Index.html:**

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8" />

<title>2-Way Live Translator</title>

<link rel="stylesheet" href="/static/style.css">

<script src="https://cdn.socket.io/4.7.2/socket.io.min.js"></script>

</head>

<body>

<div class="card">

<h1>2-Way Live Voice Translator</h1>

<div class="row">

<label>Your name:

<input id="name" value="You" />

</label>

<label>Local (mic) language:

<select id="localLang">

<option value="en">English</option>

<option value="hi">Hindi</option>

<option value="es">Spanish</option>

<option value="fr">French</option>

</select>

</label>

<label>Translate to:

<select id="targetLang">

<option value="hi">Hindi</option>

<option value="en" selected>English</option>

<option value="es">Spanish</option>

<option value="fr">French</option>

</select>

</label>

</div>

<div class="controls">

<button id="startRec">Start Listening</button>

<button id="stopRec" disabled>Stop</button>

<button id="testBroadcast">Test Broadcast</button>

</div>

<div id="log" class="log"></div>

</div>

<script src="/static/script.js"></script>

</body>

</html>

**SCRIPT.JS:**

// static/script.js

// Client: SpeechRecognition -> emit text\_message -> receive translated\_audio and play base64 audio.

// socket + id

const socket = io();

const myId = 'c-' + Math.random().toString(36).slice(2,9);

// UI elements

const nameInput = document.getElementById('name');

const localLangEl = document.getElementById('localLang');

const targetLangEl = document.getElementById('targetLang');

const startBtn = document.getElementById('startRec');

const stopBtn = document.getElementById('stopRec');

const testBtn = document.getElementById('testBroadcast');

const logEl = document.getElementById('log');

function addLog(html){

const d = document.createElement('div'); d.innerHTML = html; logEl.appendChild(d); logEl.scrollTop = logEl.scrollHeight;

}

// socket events / debug

console.log("socket object:", socket);

socket.on('connect', () => { console.log("Socket connected:", socket.id); addLog('<small class=\"small\">Socket connected</small>'); });

socket.on('connect\_error', (err) => { console.error("Socket connect\_error", err); addLog('<span style=\"color:crimson\">Socket connect error</span>'); });

// receive broadcasted translated audio

socket.on('translated\_audio', (msg) => {

try{

// ignore our own broadcast so we don't hear our own playback

if(msg.senderId === myId){

console.log("Ignoring own message");

return;

}

addLog(<small class="small">Remote:</small> ${escapeHtml(msg.translatedText)});

if(msg.audioB64){

const audio = new Audio("data:audio/mp3;base64," + msg.audioB64);

audio.play().catch(e => {

console.warn("Audio play error:", e);

addLog('<span style="color:crimson">Audio play blocked by browser (tap the page to allow autoplay)</span>');

});

}

}catch(e){ console.error(e); }

});

// SpeechRecognition

const SpeechRecognition = window.SpeechRecognition || window.webkitSpeechRecognition;

let recognition = null;

if(!SpeechRecognition){

addLog('<span style="color:crimson">Browser does not support SpeechRecognition. Use Chrome/Edge.</span>');

startBtn.disabled = true;

} else {

recognition = new SpeechRecognition();

recognition.continuous = false;

recognition.interimResults = false;

recognition.onstart = () => {

addLog('<small style="color:green">Listening...</small>');

startBtn.disabled = true;

stopBtn.disabled = false;

};

recognition.onend = () => {

addLog('<small style="color:#666">Stopped listening</small>');

startBtn.disabled = false;

stopBtn.disabled = true;

};

recognition.onerror = (e) => {

addLog('<span style="color:crimson">Recognition error: '+ escapeHtml(e.error || e.message || e) +'</span>');

console.warn("Recognition error", e);

};

recognition.onresult = (ev) => {

const last = ev.results[ev.results.length - 1];

const text = last[0].transcript.trim();

addLog(<b>You:</b> ${escapeHtml(text)});

const payload = {

text: text,

senderId: myId,

targetLang: targetLangEl.value

};

console.log("Emitting text\_message payload:", payload);

addLog('<small class="small">Sent: ' + escapeHtml(text) + '</small>');

socket.emit('text\_message', payload);

};

}

// buttons

startBtn.addEventListener('click', () => {

try{

recognition.lang = localLangEl.value || 'en';

recognition.start();

}catch(e){

addLog('<span style="color:crimson">Start error: '+ escapeHtml(e.message || e) +'</span>');

}

});

stopBtn.addEventListener('click', () => {

try{ recognition.stop(); }catch(e){}

});

// quick test broadcast (emits a sample message from this client)

testBtn.addEventListener('click', () => {

const sample = "This is a test message from " + (nameInput.value || "You");

const payload = { text: sample, senderId: myId, targetLang: targetLangEl.value };

addLog('<small class="small">Test send: '+escapeHtml(sample)+'</small>');

socket.emit('text\_message', payload);

});

// small util

function escapeHtml(s){ return (''+s).replace(/&/g,'&amp;').replace(/</g,'&lt;').replace(/>/g,'&gt;'); }

// initial instructions

addLog('<div class="small">Open this page on two devices (same network). Set each device local language to the speaker language and target language to the language the listener should hear. Press "Start Listening" and speak. Translated audio will play on the other device.</div>');

STYLE.CSS:

body{

font-family: system-ui, "Segoe UI", Roboto, Arial;

background:#f4f6f8;

display:flex;

align-items:center;

justify-content:center;

height:100vh;

margin:0;

}

.card{

background:#fff;

padding:20px;

border-radius:12px;

box-shadow:0 8px 30px rgba(0,0,0,0.08);

width:760px;

max-width:95%;

}

h1{margin:0 0 12px 0}

.row{display:flex;gap:12px;flex-wrap:wrap;margin-bottom:12px}

label{font-size:14px;color:#333;display:flex;align-items:center;gap:8px}

input,select{padding:8px;border-radius:8px;border:1px solid #ddd}

.controls{margin-bottom:12px}

button{padding:10px 14px;border-radius:8px;border:none;background:#0066ff;color:white;cursor:pointer;margin-right:8px}

button:disabled{opacity:0.5;cursor:default}

.log{background:#fbfcfd;border-radius:8px;padding:10px;height:300px;overflow:auto;border:1px solid #eee}

.small{font-size:13px;color:#666}

# app.py

# 2-way translator server: accepts text messages from clients, translates, creates in-memory TTS mp3,

# and broadcasts base64 audio + translated text to connected clients.

import sys

from io import BytesIO

import base64

from flask import Flask, render\_template

from flask\_socketio import SocketIO

# AI libs

from googletrans import Translator

from gtts import gTTS

# Optional: guard to show helpful error if dependencies missing

try:

pass

except Exception:

print("Missing packages. Run: pip install -r requirements.txt")

sys.exit(1)

app = Flask(\_name\_)

socketio = SocketIO(app, cors\_allowed\_origins="\*") # eventlet recommended

translator = Translator()

@app.route('/')

def index():

return render\_template('index.html')

@socketio.on('connect')

def on\_connect():

print("[SOCKET] client connected")

@socketio.on('disconnect')

def on\_disconnect():

print("[SOCKET] client disconnected")

@socketio.on('text\_message')

def handle\_text\_message(payload):

"""

Expected payload: { text, senderId, targetLang }

Server will:

1) translate text -> translated

2) create TTS mp3 in memory -> base64

3) emit 'translated\_audio' { senderId, translatedText, targetLang, audioB64 }

"""

try:

text = (payload.get('text') or '').strip()

sender = payload.get('senderId', '')

target = payload.get('targetLang', 'en')

except Exception as e:

print("[ERROR] malformed payload", e)

return

print(f"[RECV] sender={sender} target={target} text={repr(text)}")

if not text:

print("[RECV] empty text -> ignored")

return

# 1) Translate

try:

translated = translator.translate(text, dest=target).text

print("[TRANSLATE] ->", translated)

except Exception as e:

translated = f"[translation error] {e}"

print("[TRANSLATE ERROR]", e)

# 2) TTS -> in-memory mp3 -> base64

audio\_b64 = ''

try:

tts = gTTS(translated, lang=target)

fp = BytesIO()

tts.write\_to\_fp(fp)

fp.seek(0)

audio\_bytes = fp.read()

audio\_b64 = base64.b64encode(audio\_bytes).decode('utf-8')

print("[TTS] generated audio size bytes:", len(audio\_bytes))

except Exception as e:

audio\_b64 = ''

print("[TTS ERROR]", e)

# 3) Broadcast to all clients

try:

socketio.emit('translated\_audio', {

'senderId': sender,

'translatedText': translated,

'targetLang': target,

'audioB64': audio\_b64

})

print("[EMIT] broadcasted translated\_audio")

except Exception as e:

print("[EMIT ERROR]", e)

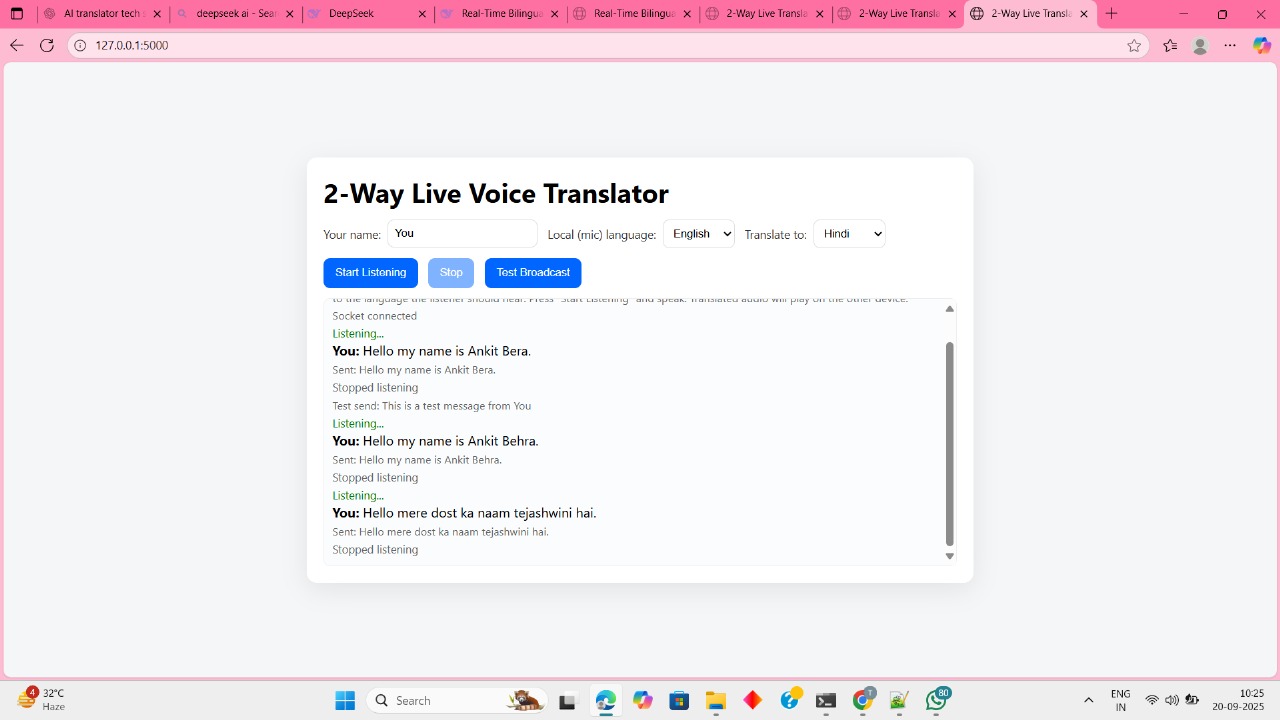
if \_name\_ == "\_main\_":

# If eventlet is installed, SocketIO will use it automatically (recommended for production/test)

print("Starting server on 0.0.0.0:5000 (use http://<PC\_IP>:5000 from other devices)")

socketio.run(app, host='0.0.0.0', port=5000, debug=True)

**photos:**

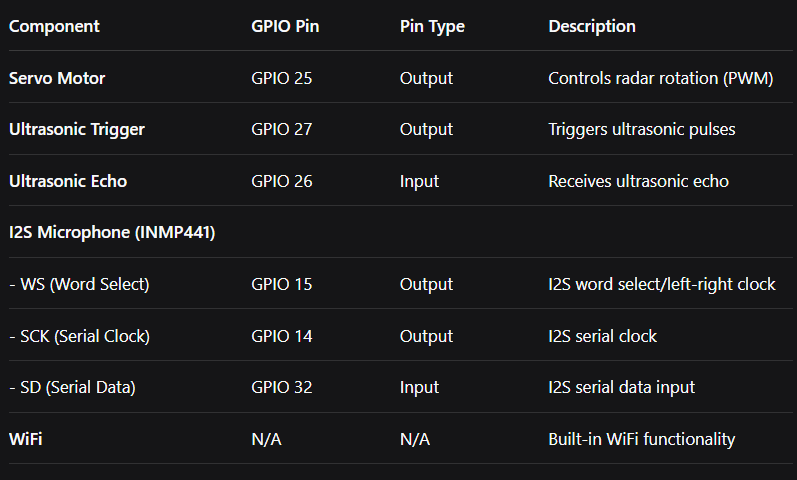


**Objective:**

The primary objective is to deny the enemy secrecy and gain a decisive information advantage by systematically collecting intelligence through technical and human means.

Core Philosophical Objective: To turn the battlefield environment (the "walls") into a source of actionable intelligence, making it impossible for the enemy to plan, move, or communicate without being detected.

**Pin Configuration**



**Code:**

#include <WiFi.h>

#include <WebServer.h>

#include <ESP32Servo.h>

#include <Ultrasonic.h>

#include <driver/i2s.h>

// WiFi credentials

const char\* ssid = "Tejaswini";

const char\* password = "password";

// Servo motor setup - UPDATED PINS

#define SERVO\_PIN 25

Servo radarServo;

const int SERVO\_MIN\_ANGLE = 0;

const int SERVO\_MAX\_ANGLE = 180;

int currentAngle = 90; // Start at center position

int scanDirection = 1;

const int SCAN\_SPEED = 2;

// Ultrasonic sensor setup - UPDATED PINS

#define TRIG\_PIN 27

#define ECHO\_PIN 26

Ultrasonic ultrasonic(TRIG\_PIN, ECHO\_PIN);

// I2S Microphone setup - INMP441

#define I2S\_WS 15

#define I2S\_SCK 14

#define I2S\_SD 32

#define I2S\_PORT I2S\_NUM\_0

// Audio recording parameters

const int SAMPLE\_RATE = 16000; // Reduced for shorter recordings

const int BITS\_PER\_SAMPLE = 16;

const int RECORD\_DURATION = 10; // 10 seconds as requested

const int BUFFER\_SIZE = 512; // Smaller buffer for shorter recordings

bool isRecording = false;

unsigned long recordingStartTime = 0;

String audioFilename = "";

// Detection parameters

const int DETECTION\_DISTANCE = 5; // 50cm range

const int DETECTION\_PAUSE = 1000; // ms to pause when object detected

// Web server

WebServer server(80);

// Detection data

String detectedObjects = "";

unsigned long lastDetectionTime = 0;

bool isPaused = false;

// Servo control variables

unsigned long lastServoMoveTime = 0;

const int SERVO\_MOVE\_DELAY = 0; // Time between servo movements (ms)

void setup() {

Serial.begin(115200);

// Initialize servo motor

radarServo.attach(SERVO\_PIN);

radarServo.write(currentAngle);

delay(1000);

// Initialize I2S microphone

setupI2S();

// Connect to WiFi

WiFi.begin(ssid, password);

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

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("Connected to WiFi");

Serial.print("IP address: ");

Serial.println(WiFi.localIP());

// Set up web server routes

server.on("/", handleRoot);

server.on("/data", handleData);

server.on("/audio", handleAudio);

server.begin();

Serial.println("HTTP server started");

Serial.println("Radar System with Microphone Initialized");

Serial.println("Recording time set to 10 seconds");

}

void setupI2S() {

i2s\_config\_t i2s\_config = {

.mode = (i2s\_mode\_t)(I2S\_MODE\_MASTER | I2S\_MODE\_RX),

.sample\_rate = SAMPLE\_RATE,

.bits\_per\_sample = I2S\_BITS\_PER\_SAMPLE\_16BIT,

.channel\_format = I2S\_CHANNEL\_FMT\_ONLY\_LEFT,

.communication\_format = I2S\_COMM\_FORMAT\_STAND\_I2S,

.intr\_alloc\_flags = ESP\_INTR\_FLAG\_LEVEL1,

.dma\_buf\_count = 4, // Reduced buffer count for shorter recordings

.dma\_buf\_len = BUFFER\_SIZE,

.use\_apll = false,

.tx\_desc\_auto\_clear = false,

.fixed\_mclk = 0

};

i2s\_pin\_config\_t pin\_config = {

.bck\_io\_num = I2S\_SCK,

.ws\_io\_num = I2S\_WS,

.data\_in\_num = I2S\_SD,

};

i2s\_driver\_install(I2S\_PORT, &i2s\_config, 0, NULL);

i2s\_set\_pin(I2S\_PORT, &pin\_config);

i2s\_set\_clk(I2S\_PORT, SAMPLE\_RATE, I2S\_BITS\_PER\_SAMPLE\_16BIT, I2S\_CHANNEL\_MONO);

Serial.println("I2S microphone initialized");

}

void startRecording() {

if (!isRecording) {

isRecording = true;

recordingStartTime = millis();

audioFilename = "/audio\_" + String(millis()) + ".raw";

Serial.println("Recording started: " + audioFilename);

}

}

void stopRecording() {

if (isRecording) {

isRecording = false;

Serial.println("Recording stopped");

}

}

void recordAudio() {

if (!isRecording) return;

static int16\_t i2s\_data[BUFFER\_SIZE] = {0};

size\_t bytes\_read = 0;

// Read audio data from I2S

i2s\_read(I2S\_PORT, &i2s\_data, BUFFER\_SIZE \* sizeof(int16\_t), &bytes\_read, portMAX\_DELAY);

// Here you would save the audio data to SPIFFS or SD card

// For simplicity, we're just simulating the recording process

// Check if recording duration has elapsed

if (millis() - recordingStartTime > (RECORD\_DURATION \* 1000)) {

stopRecording();

}

}

void moveServo() {

// Only move servo at regular intervals

if (millis() - lastServoMoveTime < SERVO\_MOVE\_DELAY) {

return;

}

lastServoMoveTime = millis();

// Move servo

currentAngle += scanDirection \* SCAN\_SPEED;

// Change direction at limits

if (currentAngle >= SERVO\_MAX\_ANGLE || currentAngle <= SERVO\_MIN\_ANGLE) {

scanDirection \*= -1;

// Ensure we stay within bounds

currentAngle = constrain(currentAngle, SERVO\_MIN\_ANGLE, SERVO\_MAX\_ANGLE);

}

radarServo.write(currentAngle);

}

void loop() {

server.handleClient();

if (isPaused) {

if (millis() - lastDetectionTime > DETECTION\_PAUSE) {

isPaused = false;

Serial.println("Resuming scan");

}

return;

}

// Handle audio recording

if (isRecording) {

recordAudio();

// Continue scanning while recording

}

// Move servo (with controlled timing)

moveServo();

// Read distance

long distance = ultrasonic.read();

// Check for objects

if (distance > 0 && distance <= DETECTION\_DISTANCE) {

// Add to detected objects

detectedObjects = "{\"angle\":" + String(currentAngle) +

",\"distance\":" + String(distance) +

",\"time\":" + String(millis()) + "}";

// Start recording when object is detected

startRecording();

// Pause scanning briefly

isPaused = true;

lastDetectionTime = millis();

}

// Small delay to prevent CPU overuse

delay(10);

}

void handleRoot() {

String html = R"=====(

<!DOCTYPE html>

<html>

<head>

<title>ESP32 Radar System</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

<style>

body {

font-family: Arial, sans-serif;

text-align: center;

background-color: #0a1929;

color: #e0f7fa;

margin: 0;

padding: 20px;

}

.container {

max-width: 600px;

margin: 0 auto;

}

.radar-container {

position: relative;

width: 500px;

height: 300px;

margin: 20px auto;

background: linear-gradient(to bottom, #001e3c, #000510);

border-radius: 10px;

border: 2px solid #00e5ff;

overflow: hidden;

}

.radar-sweep {

position: absolute;

width: 100%;

height: 100%;

background: linear-gradient(90deg, transparent 50%, rgba(0, 229, 255, 0.1) 100%);

transform-origin: center bottom;

clip-path: polygon(0% 100%, 50% 0%, 100% 100%);

}

.detection-dot {

position: absolute;

width: 12px;

height: 12px;

background: #ff3d00;

border-radius: 50%;

transform: translate(-50%, -50%);

box-shadow: 0 0 15px #ff3d00, 0 0 30px #ff3d00;

z-index: 10;

animation: pulse 1.5s infinite;

}

@keyframes pulse {

0% { opacity: 1; }

50% { opacity: 0.5; }

100% { opacity: 1; }

}

.grid-line {

position: absolute;

border: 1px solid rgba(0, 229, 255, 0.3);

border-radius: 50%;

bottom: 0;

left: 50%;

transform: translateX(-50%);

}

.grid-label {

position: absolute;

color: #00e5ff;

font-size: 12px;

font-weight: bold;

}

.angle-line {

position: absolute;

background: rgba(0, 229, 255, 0.2);

height: 150px;

width: 1px;

bottom: 0;

left: 50%;

transform-origin: bottom center;

}

.angle-label {

position: absolute;

color: #00e5ff;

font-size: 12px;

bottom: 10px;

}

.info-panel {

background: rgba(0, 30, 60, 0.8);

padding: 15px;

border-radius: 10px;

border: 1px solid #00e5ff;

margin: 20px auto;

}

.status-indicator {

display: inline-block;

width: 12px;

height: 12px;

border-radius: 50%;

margin-right: 8px;

}

.status-scanning { background: #00e5ff; box-shadow: 0 0 10px #00e5ff; }

.status-detection { background: #ff3d00; box-shadow: 0 0 10px #ff3d00; }

.status-recording { background: #ff4081; box-shadow: 0 0 10px #ff4081; }

.detection-history {

margin-top: 20px;

max-height: 150px;

overflow-y: auto;

text-align: left;

background: rgba(0, 20, 40, 0.8);

padding: 10px;

border-radius: 5px;

}

.audio-controls {

margin: 20px 0;

padding: 15px;

background: rgba(0, 30, 60, 0.8);

border-radius: 10px;

border: 1px solid #00e5ff;

}

.audio-visualizer {

width: 100%;

height: 60px;

background: rgba(0, 0, 0, 0.3);

border-radius: 5px;

margin: 10px 0;

overflow: hidden;

position: relative;

}

.audio-bar {

position: absolute;

bottom: 0;

width: 4px;

background: #00e5ff;

margin: 0 1px;

border-radius: 2px 2px 0 0;

}

</style>

</head>

<body>

<div class="container">

<h1>ESP32 Radar System with Audio Recording</h1>

<div class="info-panel">

<div>

<span id="status-icon" class="status-indicator status-scanning"></span>

<span id="status-text">Status: Scanning</span>

</div>

<div id="ip-address">IP: )=====";

html += WiFi.localIP().toString();

html += R"=====(</div>

<div id="current-reading">Current: Angle: 0°, Distance: 0.00 m</div>

<div id="detection-count">Objects detected: 0</div>

<div id="recording-status">Audio: Not recording</div>

<div id="recording-time">Recording time: 0s/10s</div>

</div>

<div class="audio-controls">

<h3>Audio Recording (10 seconds)</h3>

<div class="audio-visualizer" id="visualizer"></div>

<div id="audio-info">Microphone: INMP441 I2S (16kHz, 16-bit)</div>

<div id="recording-info">Recording starts automatically when object is detected (10 seconds)</div>

</div>

<div class="radar-container">

<div class="radar-sweep" id="sweep"></div>

<div id="detections"></div>

<!-- Range circles -->

<div class="grid-line" style="width: 200px; height: 200px;"></div>

<div class="grid-line" style="width: 300px; height: 300px;"></div>

<div class="grid-line" style="width: 400px; height: 400px;"></div>

<!-- Range labels -->

<div class="grid-label" style="left: 70%; bottom: 90px;">1m</div>

<div class="grid-label" style="left: 80%; bottom: 40px;">2m</div>

<!-- Angle lines -->

<div class="angle-line" style="transform: rotate(-90deg);"></div>

<div class="angle-line" style="transform: rotate(-45deg);"></div>

<div class="angle-line" style="transform: rotate(0deg);"></div>

<div class="angle-line" style="transform: rotate(45deg);"></div>

<div class="angle-line" style="transform: rotate(90deg);"></div>

<!-- Angle labels -->

<div class="angle-label" style="left: 50%;">0°</div>

<div class="angle-label" style="left: 25%;">45°</div>

<div class="angle-label" style="left: 75%;">135°</div>

<div class="angle-label" style="left: 5%;">90°L</div>

<div class="angle-label" style="left: 95%;">90°R</div>

</div>

<div class="detection-history">

<h3>Detection History</h3>

<div id="history-list"></div>

</div>

</div>

<script>

let detections = [];

let detectionHistory = [];

let isRecording = false;

let recordingTime = 0;

let recordingTimer = null;

function updateRadar() {

fetch('/data')

.then(response => response.json())

.then(data => {

if (data.angle !== undefined && data.distance !== undefined) {

// Add new detection

const detection = {

angle: parseInt(data.angle),

distance: parseInt(data.distance),

time: Date.now()

};

detections.push(detection);

// Add to history

detectionHistory.unshift(detection);

if (detectionHistory.length > 10) {

detectionHistory.pop();

}

// Update UI

updateStatus('DETECTION', detection);

updateHistory();

// Start recording if not already recording

if (!isRecording) {

startRecording();

}

}

// Update radar display

updateRadarDisplay();

// Remove old detections (after 5 seconds)

const now = Date.now();

detections = detections.filter(d => now - d.time < 5000);

})

.catch(error => console.error('Error:', error));

setTimeout(updateRadar, 200);

}

function updateRadarDisplay() {

const container = document.getElementById('detections');

container.innerHTML = '';

detections.forEach(detection => {

// Convert polar to Cartesian coordinates

const angle = detection.angle;

const distance = Math.min(detection.distance / 100, 2); // Convert to meters, max 2m

// Calculate position (180-degree radar)

const x = 250 + (distance \* 100) \* Math.sin((angle \* Math.PI) / 180);

const y = 300 - (distance \* 100) \* Math.cos((angle \* Math.PI) / 180);

const dot = document.createElement('div');

dot.className = 'detection-dot';

dot.style.left = x + 'px';

dot.style.top = y + 'px';

dot.title = Angle: ${angle}°, Distance: ${(detection.distance/100).toFixed(2)} m;

container.appendChild(dot);

});

// Update detection count

document.getElementById('detection-count').textContent =

Objects detected: ${detections.length};

}

function updateStatus(status, detection = null) {

const icon = document.getElementById('status-icon');

const text = document.getElementById('status-text');

const reading = document.getElementById('current-reading');

const recordingStatus = document.getElementById('recording-status');

if (status === 'DETECTION' && detection) {

icon.className = 'status-indicator status-detection';

text.textContent = 'Status: Object Detected!';

reading.textContent = Current: Angle: ${detection.angle}°, Distance: ${(detection.distance/100).toFixed(2)} m;

// Return to scanning after 2 seconds

setTimeout(() => {

if (!isRecording) {

icon.className = 'status-indicator status-scanning';

text.textContent = 'Status: Scanning';

}

}, 2000);

} else if (status === 'RECORDING') {

icon.className = 'status-indicator status-recording';

text.textContent = 'Status: Recording Audio';

recordingStatus.textContent = 'Audio: Recording (' + recordingTime + 's/10s)';

} else {

icon.className = 'status-indicator status-scanning';

text.textContent = 'Status: Scanning';

recordingStatus.textContent = 'Audio: Not recording';

}

}

function updateHistory() {

const historyList = document.getElementById('history-list');

historyList.innerHTML = '';

detectionHistory.forEach(detection => {

const time = new Date(detection.time).toLocaleTimeString();

const item = document.createElement('div');

item.textContent = ${time} - Angle: ${detection.angle}°, Distance: ${(detection.distance/100).toFixed(2)} m;

item.style.margin = '5px 0';

item.style.padding = '5px';

item.style.background = 'rgba(255, 61, 0, 0.1)';

item.style.borderRadius = '3px';

historyList.appendChild(item);

});

}

function startRecording() {

isRecording = true;

recordingTime = 0;

updateStatus('RECORDING');

// Update recording time every second

recordingTimer = setInterval(() => {

recordingTime++;

document.getElementById('recording-time').textContent = Recording time: ${recordingTime}s/10s;

if (recordingTime >= 10) { // 10 seconds

stopRecording();

}

}, 1000);

// Simulate audio visualization

simulateAudioVisualization();

}

function stopRecording() {

isRecording = false;

clearInterval(recordingTimer);

updateStatus('SCANNING');

document.getElementById('recording-time').textContent = 'Recording time: 0s/10s';

}

function simulateAudioVisualization() {

const visualizer = document.getElementById('visualizer');

visualizer.innerHTML = '';

for (let i = 0; i < 50; i++) {

const bar = document.createElement('div');

bar.className = 'audio-bar';

bar.style.left = (i \* 10) + 'px';

bar.style.height = Math.random() \* 50 + 5 + 'px';

visualizer.appendChild(bar);

}

if (isRecording) {

setTimeout(simulateAudioVisualization, 100);

}

}

function animateSweep() {

const sweep = document.getElementById('sweep');

const now = Date.now();

const degrees = (now / 20) % 360;

sweep.style.transform = rotate(${degrees}deg);

requestAnimationFrame(animateSweep);

}

// Initialize

updateRadar();

animateSweep();

</script>

</body>

</html>

)=====";

server.send(200, "text/html", html);

}

void handleData() {

if (detectedObjects != "") {

server.send(200, "application/json", detectedObjects);

detectedObjects = "";

} else {

server.send(200, "application/json", "{}");

}

}

void handleAudio() {

// This would handle audio file downloads

server.send(200, "text/plain", "Audio recording functionality would be implemented here");

}

**Photo:**

