**Updated code with GA(Genetic algorithm) to get optimized parking slot:**

#include <ESP8266WiFi.h>

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

#include <SPI.h>

#include <MFRC522.h>

// RFID Module Pins

#define SS\_PIN 15

#define RST\_PIN 16

MFRC522 rfid(SS\_PIN, RST\_PIN);

// Define total parking slots

#define TOTAL\_SLOTS 5

#define POPULATION\_SIZE 10

#define MUTATION\_RATE 0.2

#define GENERATIONS 50

#define DISTANCE\_THRESHOLD 40  // If distance < 40cm, spot is occupied

// Ultrasonic sensor pins (for each slot)

const int trigPins[TOTAL\_SLOTS] = {5, 6, 7, 8, 9};

const int echoPins[TOTAL\_SLOTS] = {10, 11, 12, 13, 14};

// Define slot properties

struct ParkingSlot {

    int id;

    int size;

    int distance;

    bool occupied;

};

// Parking slots

ParkingSlot slots[TOTAL\_SLOTS] = {

    {1, 2, 5, false},

    {2, 3, 8, false},

    {3, 1, 3, false},

    {4, 4, 10, false},

    {5, 2, 6, false}

};

// Vehicle database

struct Vehicle {

    String rfidTag;

    int type; // 1=Compact, 2=Sedan, 3=SUV, 4=Truck

};

Vehicle vehicleDB[] = {

    {"123A4B", 1}, // Compact

    {"567C8D", 2}, // Sedan

    {"9EFG12", 3}, // SUV

    {"34HIJK", 4}  // Truck

};

// GA population

struct Chromosome {

    int slotIndex;

    float fitness;

};

// Initialize LCD

LiquidCrystal\_I2C lcd(0x27, 16, 2);

// Function to check if a slot is occupied using an ultrasonic sensor

bool isSlotOccupied(int slotIndex) {

    long duration;

    int distance;

    digitalWrite(trigPins[slotIndex], LOW);

    delayMicroseconds(2);

    digitalWrite(trigPins[slotIndex], HIGH);

    delayMicroseconds(10);

    digitalWrite(trigPins[slotIndex], LOW);

    duration = pulseIn(echoPins[slotIndex], HIGH);

    distance = duration \* 0.034 / 2;  // Convert to cm

    return (distance < DISTANCE\_THRESHOLD); // True if distance is less than threshold

}

// Function to update slot occupancy status

void updateSlotOccupancy() {

    for (int i = 0; i < TOTAL\_SLOTS; i++) {

        slots[i].occupied = isSlotOccupied(i);

    }

}

// Function to get vehicle type from RFID tag

int getVehicleType(String rfidTag) {

    for (int i = 0; i < sizeof(vehicleDB) / sizeof(vehicleDB[0]); i++) {

        if (vehicleDB[i].rfidTag == rfidTag) {

            return vehicleDB[i].type;

        }

    }

    return -1; // Unknown vehicle

}

// Fitness function

float calculateFitness(Chromosome individual, int vehicleSize) {

    int idx = individual.slotIndex;

    float fitness = 100 - (slots[idx].distance \* 2);

    if (slots[idx].size < vehicleSize) fitness -= 30;

    if (slots[idx].occupied) fitness -= 50;

    return fitness;

}

// Genetic Algorithm main function (uses only available slots)

int geneticAlgorithm(int vehicleSize) {

    Chromosome population[POPULATION\_SIZE];

    int availableSlots[TOTAL\_SLOTS];

    int availableCount = 0;

    // Store available slots

    for (int i = 0; i < TOTAL\_SLOTS; i++) {

        if (!slots[i].occupied) {

            availableSlots[availableCount++] = i;

        }

    }

    if (availableCount == 0) return -1; // No available slots

    // Initialize population

    for (int i = 0; i < POPULATION\_SIZE; i++) {

        int randomIndex = random(availableCount);

        population[i].slotIndex = availableSlots[randomIndex];

        population[i].fitness = calculateFitness(population[i], vehicleSize);

    }

    // Run GA for several generations

    for (int gen = 0; gen < GENERATIONS; gen++) {

        Chromosome best1 = population[0], best2 = population[0];

        for (int i = 1; i < POPULATION\_SIZE; i++) {

            if (population[i].fitness > best1.fitness) {

                best2 = best1;

                best1 = population[i];

            } else if (population[i].fitness > best2.fitness) {

                best2 = population[i];

            }

        }

        Chromosome newChild;

        newChild.slotIndex = (best1.slotIndex + best2.slotIndex) / 2;

        newChild.fitness = calculateFitness(newChild, vehicleSize);

        if (random(100) < MUTATION\_RATE \* 100) {

            newChild.slotIndex = availableSlots[random(availableCount)];

            newChild.fitness = calculateFitness(newChild, vehicleSize);

        }

        population[POPULATION\_SIZE - 1] = newChild;

    }

    return population[0].slotIndex;

}

// Setup

void setup() {

    Serial.begin(115200);

    SPI.begin();

    rfid.PCD\_Init();

    lcd.begin();

    lcd.backlight();

    lcd.setCursor(0, 0);

    lcd.print("Smart Parking");

    // Initialize ultrasonic sensors

    for (int i = 0; i < TOTAL\_SLOTS; i++) {

        pinMode(trigPins[i], OUTPUT);

        pinMode(echoPins[i], INPUT);

    }

}

// Main loop

void loop() {

    updateSlotOccupancy(); // Update slot occupancy using sensors

    if (rfid.PICC\_IsNewCardPresent() && rfid.PICC\_ReadCardSerial()) {

        String rfidID = "";

        for (byte i = 0; i < rfid.uid.size; i++) {

            rfidID += String(rfid.uid.uidByte[i], HEX);

        }

        Serial.print("Detected Vehicle ID: ");

        Serial.println(rfidID);

        int vehicleType = getVehicleType(rfidID);

        if (vehicleType == -1) {

            Serial.println("Unknown vehicle!");

            lcd.clear();

            lcd.setCursor(0, 0);

            lcd.print("Unknown Vehicle");

        } else {

            int bestSlot = geneticAlgorithm(vehicleType);

            lcd.clear();

            if (bestSlot == -1) {

                lcd.setCursor(0, 0);

                lcd.print("No Slot Available");

                Serial.println("No Slot Available!");

            } else {

                lcd.setCursor(0, 0);

                lcd.print("Best Slot: ");

                lcd.print(slots[bestSlot].id);

                Serial.print("Optimized Parking Slot: ");

                Serial.println(slots[bestSlot].id);

            }

        }

        rfid.PICC\_HaltA();

    }

    delay(5000);

}