Smart Waste Bin with ESP32 and YOLOv5

Overview

This project is a smart waste bin system built using ESP32, sensors, and machine learning. The system detects plastic waste using a YOLOv5 model running on a Raspberry Pi. When plastic is detected, the servo-controlled door opens, and the data is sent to Firebase for real-time tracking.

Components

ESP32: Handles communication with sensors and controls the servo motor.

YOLOv5: Detects plastic waste using a trained machine learning model on Raspberry Pi.

Firebase: Stores and updates the waste detection data in real-time.

Servo Motors: Control the door mechanism for sorting waste.

Software Setup

Install Libraries:

FirebaseESP32 library for Firebase communication

Servo library for motor control

Any other libraries for sensor integration

Upload Code to ESP32:

Use Arduino IDE to upload the code to your ESP32

Ensure the correct board and port are selected in Arduino IDE

Firebase Configuration:

Set up a Firebase project and obtain your credentials (API key, database URL, etc.)

Add Firebase configuration details to the code

YOLOv5 Model:

Train a YOLOv5 model to detect plastic waste (you can use pre-trained models or custom data)

Deploy the model on the Raspberry Pi

How It Works

Object Detection: The YOLOv5 model detects plastic waste and sends a signal to the ESP32.

Servo Control: The ESP32 processes the signal and moves the servo motor to open or close the door based on the detection result.

Firebase Update: The system updates Firebase with the type of waste detected (plastic or non-plastic) for tracking purposes.

Running the Project

Power up the ESP32 and Raspberry Pi.

Open the smart bin and place a plastic item near the sensor.

The servo motor should open if plastic is detected and close otherwise

The Firebase database will update the detection status in real-time.

```
#include <WiFi.h>
#include <WebSocketsServer.h>
#include <MFRC522.h>
#include <SPI.h>
#include <Firebase_ESP_Client.h>
#include <ESP32Servo.h>

// Firebase Credentials
#define FIREBASE_PROJECT_ID "your_project_id"
#define FIREBASE_API_KEY "your_api_key"
#define USER_EMAIL "your_email"
#define USER_PASSWORD "your_password"

// WiFi Credentials
const char* ssid = "your_ssid";
const char* password = "your_wifi_password";

// NFC Module Setup
#define SS_PIN  21
#define RST_PIN 22
MFRC522 mfrc522(SS_PIN, RST_PIN);

// WebSocket Setup
WebSocketsServer webSocket(81);

// Firebase Setup
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;

// Servo Setup
Servo segregationServo;
Servo doorServo;
#define SEGREGATION_SERVO_PIN 15
#define DOOR_SERVO_PIN 13

// Ultrasonic Sensor Setup
#define TRIG_PIN 2  // Trigger Pin
#define ECHO_PIN 4  // Echo Pin

#define BUZZER_PIN 27

#define BIN_EMPTY_CM 30  // Distance when bin is empty
```

```
#define BIN_FULL_CM 5    // Distance when bin is full

// Raspberry Pi GPIO Inputs
#define RASPBERRY_PI_PIN1 33  // Plastic detected signal
#define RASPBERRY_PI_PIN2 32  // Non-plastic detected signal

MFRC522::MIFARE_Key key;

// Timer for Firestore checks
unsigned long lastFirestoreCheck = 0;
const unsigned long firestoreCheckInterval = 5000;
unsigned long doorOpenTime = 0;
bool doorOpen = false;

void setup() {
    Serial.begin(9600);
    WiFi.begin(ssid, password);
    SPI.begin();
    mfrc522.PCD_Init();

    pinMode(TRIG_PIN, OUTPUT);
    pinMode(ECHO_PIN, INPUT);
    pinMode(RASPBERRY_PI_PIN1, INPUT_PULLDOWN);
    pinMode(RASPBERRY_PI_PIN2,  INPUT_PULLDOWN);
    pinMode(BUZZER_PIN, OUTPUT);

    byte customKey[6] = {0xD3, 0xF7, 0xD3, 0xF7, 0xD3, 0xF7};
    memcpy(key.keyByte, customKey, 6);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nWiFi Connected");
    Serial.print("ESP32 IP Address: ");
    Serial.println(WiFi.localIP());

    config.api_key = FIREBASE_API_KEY;
    auth.user.email = USER_EMAIL;
    auth.user.password = USER_PASSWORD;
    Firebase.begin(&config, &auth);
    Firebase.reconnectWiFi(true);
    updateFirestoreIP();

    segregationServo.attach(SEGREGATION_SERVO_PIN);
```

```
    segregationServo.write(117);
    doorServo.attach(DOOR_SERVO_PIN);
    doorServo.write(150);

    webSocket.begin();
    webSocket.onEvent(webSocketEvent);
    Serial.println("Setup Complete");
    webSocket.loop();
}

void loop() {
    checkNFC();
    checkPlasticDetection();
    webSocket.loop();
    if (millis() - lastFirestoreCheck >= firestoreCheckInterval) {
        lastFirestoreCheck = millis();
        checkFirestoreFlag();
        updateGarbageLevel();  // Update Firebase every 5s
    }

    if (doorOpen && millis() - doorOpenTime >= 5000) {
        Serial.println("Closing door");
        doorServo.write(150);
        doorOpen = false;
    }
    webSocket.loop();
}

void updateFirestoreIP() {
    String ipAddress = WiFi.localIP().toString();
    String documentPath = "bin/bin1";
    String jsonData = "{\"fields\": {\"esp_ip\": {\"stringValue\": \"" +
ipAddress + "\"}}}";
    Firebase.Firestore.patchDocument(&fbdo, FIREBASE_PROJECT_ID, "",
documentPath.c_str(), jsonData.c_str(), "esp_ip");
    Serial.println("Updated Firestore IP");
}

void checkNFC() {
    if (mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial()) {
        Serial.println("\nNFC Card Detected!");

        digitalWrite(BUZZER_PIN, HIGH);
        delay(200);  // Beep duration
        digitalWrite(BUZZER_PIN, LOW);
```

```
        String email = readNFCData(5) + readNFCData(6);  // Read email from
Blocks 5 & 6
        String password = readNFCData(9) + readNFCData(10);  // Read password
from Blocks 9 & 10

        email.trim();
        password.trim();

        if (email.length() > 0 && password.length() > 0) {
            String json = "{\"email\":\"" + email + "\", \"password\":\"" +
password + "\"}";
            Serial.println("Sending NFC Data: " + json);

            // Send data to WebSocket clients
            webSocket.broadcastTXT(json);
        } else {
            Serial.println("Error: Could not read email/password from NFC
card!");
        }

        mfrc522.PICC_HaltA();
        mfrc522.PCD_StopCrypto1();

        delay(1000);  // Prevent immediate re-scanning
    }
}

String readNFCData(byte block) {
    byte buffer[18];  // Buffer to store read data
    byte bufferSize = sizeof(buffer);

    byte sector = block / 4 * 4;
    MFRC522::StatusCode status = mfrc522.PCD_Authenticate(
        MFRC522::PICC_CMD_MF_AUTH_KEY_A, sector, &key, &(mfrc522.uid)
    );

    if (status != MFRC522::STATUS_OK) {
        Serial.print("Authentication failed for block ");
        Serial.print(block);
        Serial.print(": ");
        Serial.println(mfrc522.GetStatusCodeName(status));
        return "";
    }
```

```
    status = mfrc522.MIFARE_Read(block, buffer, &bufferSize);
    if (status != MFRC522::STATUS_OK) {
        Serial.println("Failed to read NFC data");
        return "";
    }

    String data = "";
    for (int i = 0; i < 16; i++) {
        if (buffer[i] == 0x00) break;
        data += (char)buffer[i];
    }

    return data;
}

void checkFirestoreFlag() {
    String documentPath = "bin/bin1";
    if (Firebase.Firestore.getDocument(&fbdo, FIREBASE_PROJECT_ID, "",
documentPath.c_str())) {
        Serial.println("Firestore Data Retrieved");
        processFirestoreData(fbdo.payload().c_str());
    } else {
        Serial.println("Failed to retrieve Firestore data");
        Serial.println(fbdo.errorReason());
    }
}

void processFirestoreData(const char* json) {
    FirebaseJson payload;
    payload.setJsonData(json);
    FirebaseJsonData flagData;
    payload.get(flagData, "fields/flag/integerValue");

    if (flagData.success) {
        int flag = flagData.intValue;
        Serial.print("Flag Value: ");
        Serial.println(flag);

        if (flag == 1) {
            Serial.println("Opening Door...");
            doorServo.write(0);
            doorOpen = true;
            doorOpenTime = millis();  // Start door open timer

            FirebaseJson content;
```

```
            content.set("fields/flag/integerValue", 0);
            Firebase.Firestore.patchDocument(&fbdo, FIREBASE_PROJECT_ID, "",
"bin/bin1", content.raw(), "flag");
        }
    } else {
        Serial.println("Failed to parse flag value");
    }
}

void checkPlasticDetection() {
    static enum {IDLE, WAITING, CHECKING, RESETTING} state = IDLE;
    static unsigned long detectStartTime = 0;
    static unsigned long doorOpenTime = 0;
    static bool objectPreviouslyDetected = false;

    static unsigned long lastCheckTime = 0;
    const unsigned long checkInterval = 100;  // Check every 100ms

    if (millis() - lastCheckTime < checkInterval) return;
    lastCheckTime = millis();

    // Measure distance using ultrasonic sensor
    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 * 0.0343 / 2;

    switch (state) {
        case IDLE:
            if (distance <= 22 && !objectPreviouslyDetected) {
                Serial.println("Object detected! Waiting 6s to read
classification...");
                detectStartTime = millis();
                state = WAITING;
                objectPreviouslyDetected = true;
            }
            break;

        case WAITING:
            if (millis() - detectStartTime >= 8000) {
                state = CHECKING;
            }
```

```
            break;

        case CHECKING: {
            int plasticSignal = digitalRead(RASPBERRY_PI_PIN1);
            int nonPlasticSignal = digitalRead(RASPBERRY_PI_PIN2);
            if (plasticSignal == HIGH) {
                Serial.println("Plastic detected! Opening door...");
                segregationServo.write(90); // Open to plastic position
                doorOpenTime = millis();

                FirebaseJson content;
                content.set("fields/flag/integerValue", 2);
                Firebase.Firestore.patchDocument(&fbdo, FIREBASE_PROJECT_ID, "",
"bin/bin1", content.raw(), "flag");

                state = RESETTING;
            } else if (nonPlasticSignal == HIGH) {
                Serial.println("Non-plastic detected! Opening door...");
                segregationServo.write(0); // Open to non-plastic position
                doorOpenTime = millis();

                FirebaseJson content;
                content.set("fields/flag/integerValue", 3);
                Firebase.Firestore.patchDocument(&fbdo, FIREBASE_PROJECT_ID, "",
"bin/bin1", content.raw(), "flag");

                state = RESETTING;
            } else {
                Serial.println("No signal detected. Resetting...");
                state = IDLE;
                objectPreviouslyDetected = false;
            }
            break;
        }

        case RESETTING:
            if (millis() - doorOpenTime >= 2000) { // Door stays open for 2s
                Serial.println("Closing door...");
                segregationServo.write(117); // Back to center
                state = IDLE;
                objectPreviouslyDetected = false;
            }
            break;
    }
```

```
    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");
}

void moveServo(int angle) {
    segregationServo.write(angle);
}

void webSocketEvent(uint8_t num, WStype_t type, uint8_t* payload, size_t length)
{
    switch (type) {
        case WStype_CONNECTED:
            Serial.println("WebSocket Client Connected!");
            break;
        case WStype_DISCONNECTED:
            Serial.println("WebSocket Client Disconnected!");
            break;
        case WStype_TEXT:
            Serial.print("Message Received: ");
            Serial.println((char*)payload);
            break;
    }
}

void updateGarbageLevel() {
    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;  // Convert to cm

    int garbagePercentage = map(distance, BIN_EMPTY_CM, BIN_FULL_CM, 0, 100);
    garbagePercentage = constrain(garbagePercentage, 0, 100);

    Serial.print("Garbage Level: ");
    Serial.print(garbagePercentage);
    Serial.println("%");

    String documentPath = "bin/bin1";
    String jsonData = "{\"fields\": {\"garbageLevel\": {\"integerValue\": " +
String(garbagePercentage) + "}}}";
```

```
    if (Firebase.Firestore.patchDocument(&fbdo, FIREBASE_PROJECT_ID, "",
documentPath.c_str(), jsonData.c_str(), "garbageLevel")) {
        Serial.println("Garbage Level Updated in Firebase");
    } else {
        Serial.println("Failed to update garbage level");
        Serial.println(fbdo.errorReason());
    }
}
```