EMBEDDED SYSTEM WITH IOT

EXPERIMENT NO. 2

Home Appliance Automation using IoT

Name: NAVARRO, ROD GERYK C.

Course/Section: CPE162P-4/E01

Date of Performance: 05/10/2025

Date of Submission: 05/12/2025

<u>CYREL O. MANLISES, PH.D.</u>

Instructor

DISCUSSION

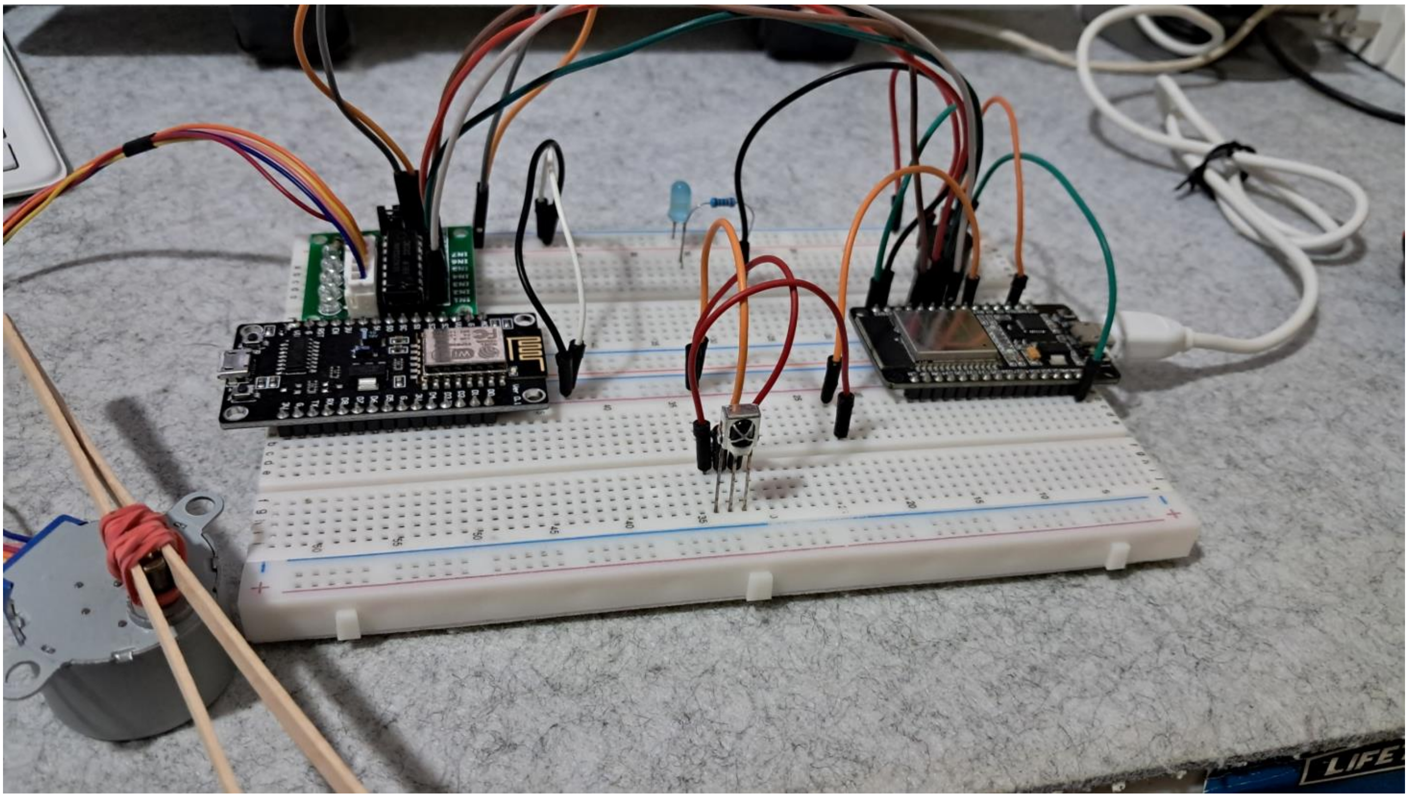

FIGURE 1: THE CONNECTION OF THE LED, STEPPER MOTOR, MOTOR MODULE, IR SENSOR, AND BUZZER, TO THE ESP32 DEV MODULE

This experiment is about building an automated appliance controller using the ESP32 dev module. the ESP32 is a low-cost, low-power microcontroller with built-in Wi-Fi and Bluetooth, which makes it perfect for internet of things projects. Its Wi-Fi feature allows it to send data over the internet. The first step in the experiment was to set up the physical circuit and connect all the components properly. The main components used were a stepper motor with its motor driver (acting as an electric fan), an LED (acting as a light bulb), a buzzer (acting as a house alarm), an IR sensor (which receives signals from a remote), and an IR remote (used to turn the appliances on and off). The ESP32 served as the main microcontroller that controls everything. a 220-ohm resistor was added to the LED to protect it from too much current. One of the goals of this *project was to send data to ThingSpeak for monitoring and analysis. All appliances in this setup were designed to behave like real household appliances and could be wirelessly controlled using the IR remote with the help of the IR sensor.*

Navarro_EXP2_CPE162P.ino

```
#include <WiFi.h>
#include <ThingSpeak.h>
#include <IRremote.hpp>

// Wifi credentials
const char* ssid = "PLDTHOMEFIBRt6ucX";
const char* password = "PLDTWIFItLK72";

// ThingSpeak setup
WiFiClient client;
long myChannelNumber = 2956949;
const char myWriteAPIKey[] = "6N5DF48YEBETNPWH";

//IR device PinsS
const int irRemotePin = 16;
const int lightLed = 22;
const int buzzAlarm = 23;
int portIN1 = 5;
int portIN2 = 18;
int portIN3 = 19;
int portIN4 = 21;

bool motorState = false;
int lightStatus = 0;
int buzzerStatus = 0;
int fanStatus = 0;

// thingspeak timer
unsigned long lastUpdate = 0;
const unsigned long updateInterval = 15000;

```

FIGURE 2: THE INITIALIZATION OF VARIABLES AND COMPONENTS

In this experiment, I started by including the needed libraries: WiFi.h to connect the ESP32 to the internet, ThingSpeak.h to send data to the cloud, and irremote.hpp so the ESP32 can read signals from the IR remote. I also created variables for Wi-Fi connection, the ThingSpeak channel, and the appliances like the led (light), buzzer (alarm), and stepper motor (electric fan). I used bool and int to keep track of whether these devices are on or off. These variables are important because they help the code *understand the current state of each appliance, and they are also sent to ThingSpeak for remote monitoring.*

Navarro_EXP2_CPE162P.ino

```
void setup() {
  Serial.begin(9600);
  delay(1000);

  pinMode(lightLed, OUTPUT);
  pinMode(buzzAlarm, OUTPUT);
  pinMode(portIN1, OUTPUT);
  pinMode(portIN2, OUTPUT);
  pinMode(portIN3, OUTPUT);
  pinMode(portIN4, OUTPUT);

  IrReceiver.begin(irRemotePin, ENABLE_LED_FEEDBACK); // start IR receiver

  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED){
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  ThingSpeak.begin(client);

}
```

FIGURE 3: THE SETUP OF THE COMPONENTS

Inside the setup() function, I used pinMode() to define which pins are for input or output. For example, the led and buzzer are outputs, so I used pinMode(lightled, output);. I also started the IR receiver so that the ESP32 can read signals from the IR remote. then I connected the ESP32 to Wi-Fi using the credentials I provided. If the connection was successful, it showed the IP address in the serial monitor. Finally, I started the ThingSpeak connection with ThingSpeak.begin(client); so I could later send data to my online dashboard.

Navarro_EXP2_CPE162P.ino

```cpp
void loop() {
  // handle IR input
  if (IrReceiver.decode()){
   Serial.print("Received IR code: ");
   Serial.println(IrReceiver.decodedIRData.decodedRawData, HEX);
   IrReceiver.printIRResultShort(&Serial);
   executeCommand(IrReceiver.decodedIRData.decodedRawData);
   IrReceiver.resume();
  }

   // Stepper motor Control
   if(motorState){
     stepSequence(1,0,0,0);
     delay(2);
     stepSequence(0,1,0,0);
     delay(2);
     stepSequence(0,0,1,0);
     delay(2);
     stepSequence(0,0,0,1);
     delay(2);
   } else{
     stepSequence(0,0,0,0);
   }

   //ThinfSpeak update every 15 seconds
   if (millis() - lastUpdate >= updateInterval) {
     lastUpdate = millis();

     lightStatus = digitalRead(lightLed);
     buzzerStatus = digitalRead(buzzAlarm);
     fanStatus = motorState ? 1 : 0;

     ThingSpeak.setField(1, lightStatus);
     ThingSpeak.setField(2, buzzerStatus);
     ThingSpeak.setField(3, fanStatus);
```

Navarro_EXP2_CPE162P.ino

```cpp
}

void executeCommand(unsigned long command) {
  switch (command) {
    case 0xF30CFF00: Serial.println("1: Toggle Motor"); moveStepper(); break;
    case 0xF708FF00: Serial.println("4: Toggle Light"); lightControl(); break;
    case 0xE31CFF00: Serial.println("5: Toggle Buzzer"); buzzerControl(); break;
    case 0xB54AFF00: Serial.println("9: Reset All"); resetAll(); break;
    default: Serial.println("Unrecognized command."); break;
  }
}

void moveStepper(){
  motorState = !motorState;
}

void lightControl(){
  digitalWrite(lightLed, !digitalRead(lightLed));
  delay(50);
}

void buzzerControl(){
  digitalWrite(buzzAlarm, !digitalRead(buzzAlarm));
  delay(50);
}

void stepSequence(int in1, int in2, int in3, int in4){
  digitalWrite(portIN1, in1);
  digitalWrite(portIN2, in2);
  digitalWrite(portIN3, in3);
  digitalWrite(portIN4, in4);
}

void resetAll(){
  motorState = false;
  digitalWrite(lightLed, LOW);
  digitalWrite(buzzAlarm, LOW);
}
```

FIGURE 4: LOOP FUNCTION AND OTHER NECESSARY FUNCTIONS

The loop() function is where the real action happens. first, the ESP32 checks if the IR receiver got a signal. If it did, the code reads it and then decides what to do, like turning the motor (fan), light, or buzzer on or off. The motor uses a step sequence with four digital pins to rotate in small steps. I also added a timer using millis() so the ESP32 sends data to ThingSpeak every 15 seconds. It reads the status of each appliance (on or off) and sends it to fields in my ThingSpeak channel for monitoring. This is useful if I want to check which appliances are on even when I m not at home.

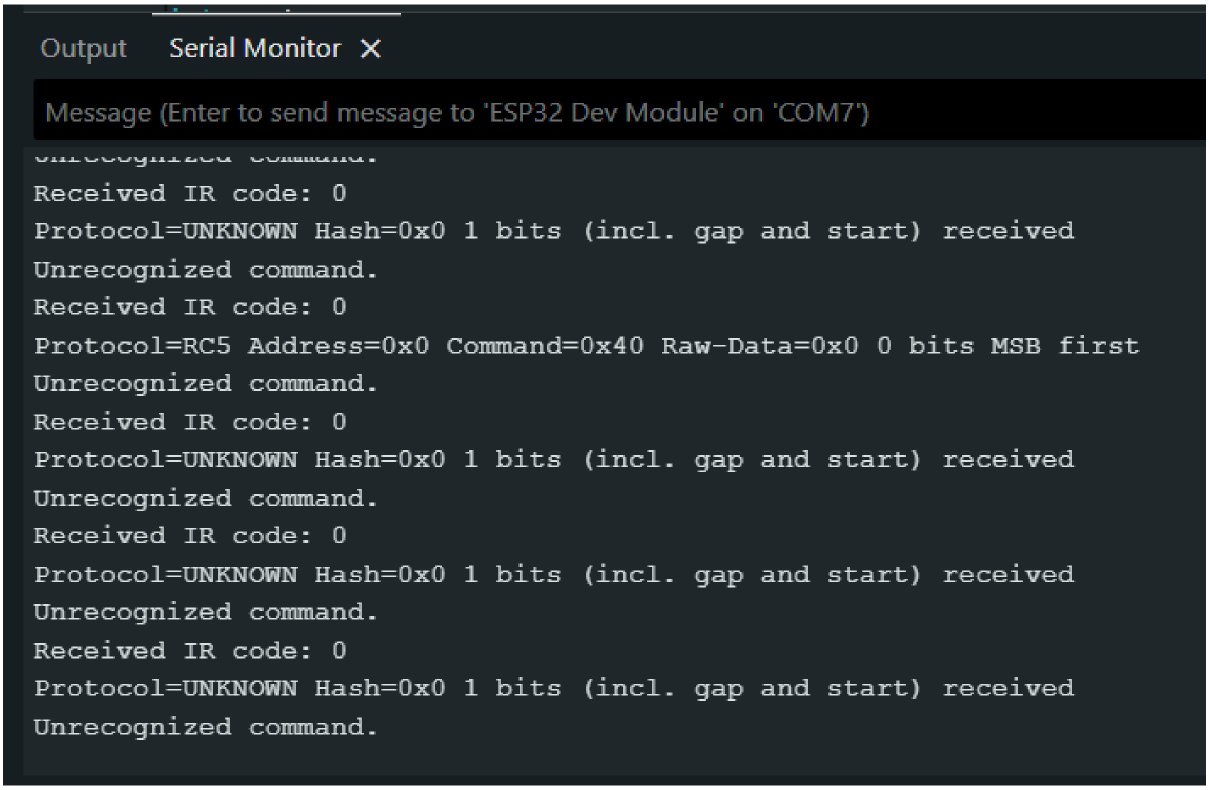

FIGURE 5: FIXING THE ERRORS

While testing the code, one issue I encountered was incorrect LED behavior. At first, some LEDs didnt light up because I forgot to match the physical wiring with the pin numbers in the code. After double-checking both the code and the breadboard, I fixed the pin assignments. I also noticed I had a typo in a comment which I corrected to make the code easier to understand.

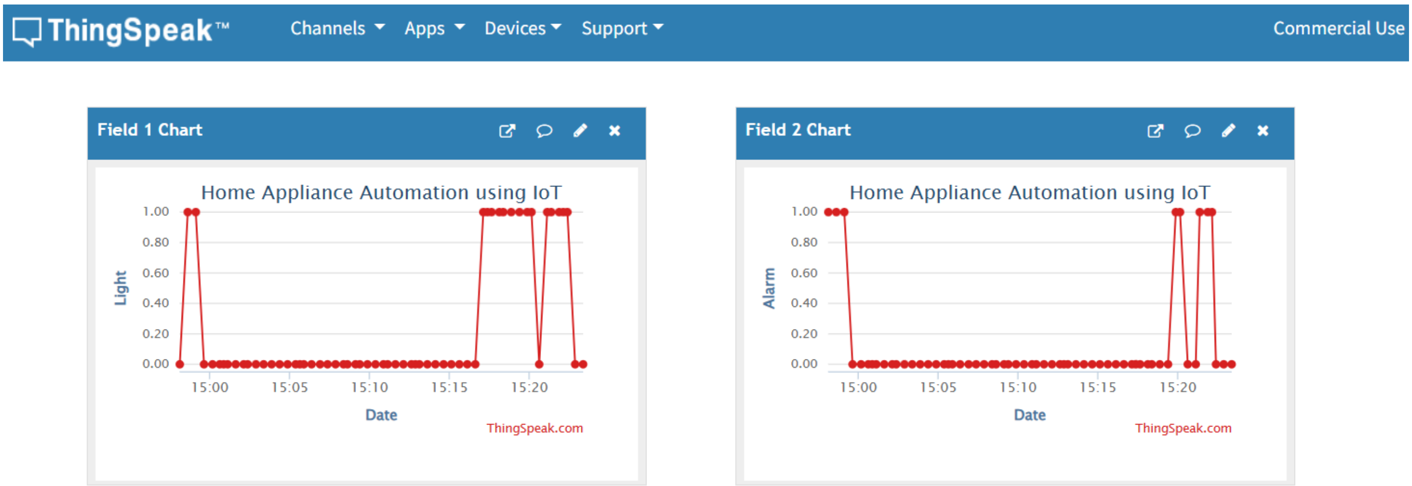

FIGURE 6: THINGSPEAK GRAPH INTERPRETATION: LIGHT AND ALARM GRAPHS

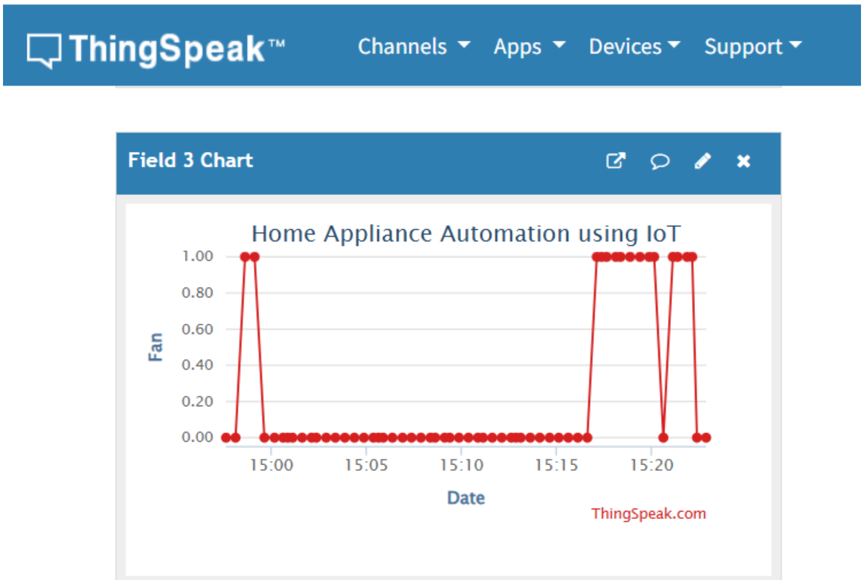

FIGURE 7: THINGSPEAK GRAPH INTERPRETATION: FAN GRAPH

In ThingSpeak, I created graphs for each appliance: light, buzzer, and fan. Each graph shows when the appliance was turned on (value = 1) and off (value = 0). For example, when I press button 4 on the remote to toggle the light, the graph on field 1 updates with a 1 or 0, depending on the light s state. This helps me track how often each appliance is used and when. It feels like having a smart home dashboard where I can monitor my devices in real time.

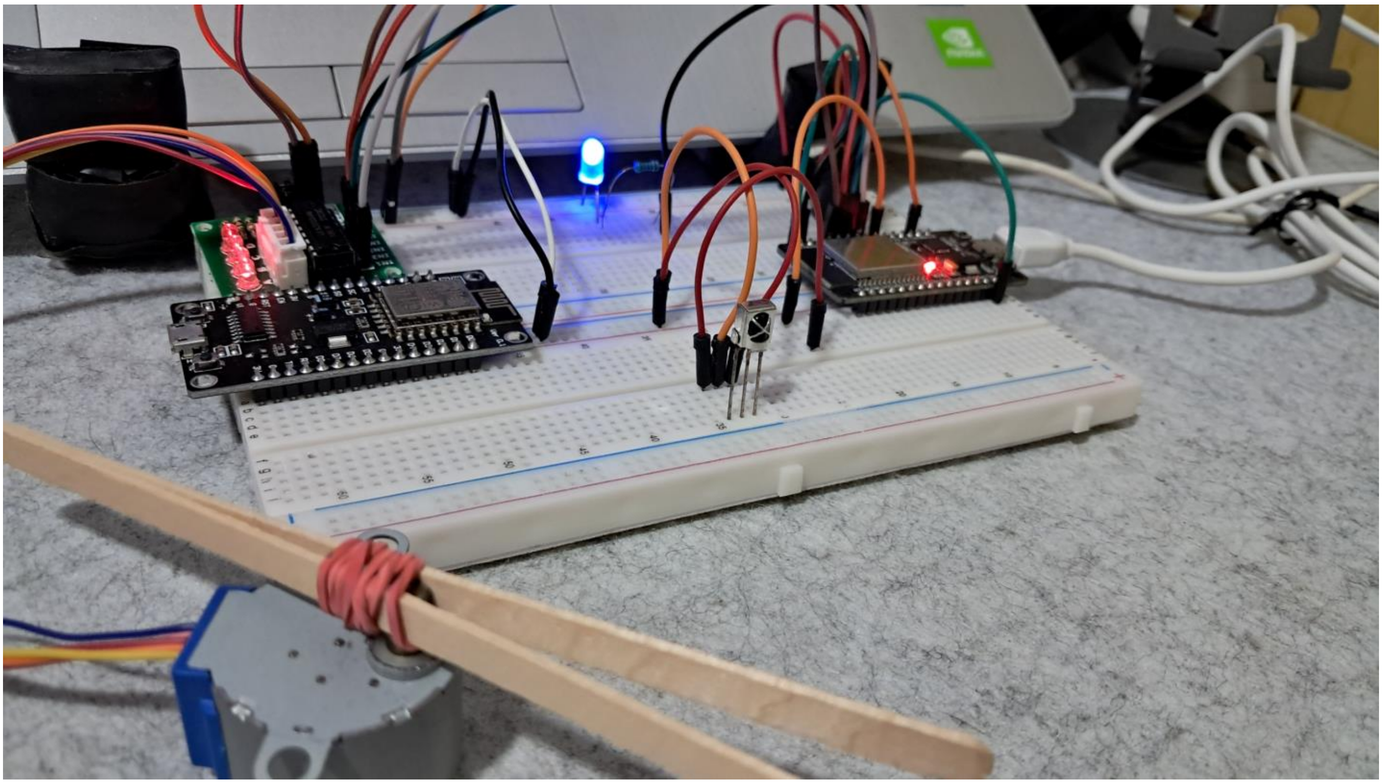

FIGURE 8: WORKING PROTOTYPE: ALL APPLIANCES ARE TURNED ON - LIGHT, BUZZER, AND ALARM

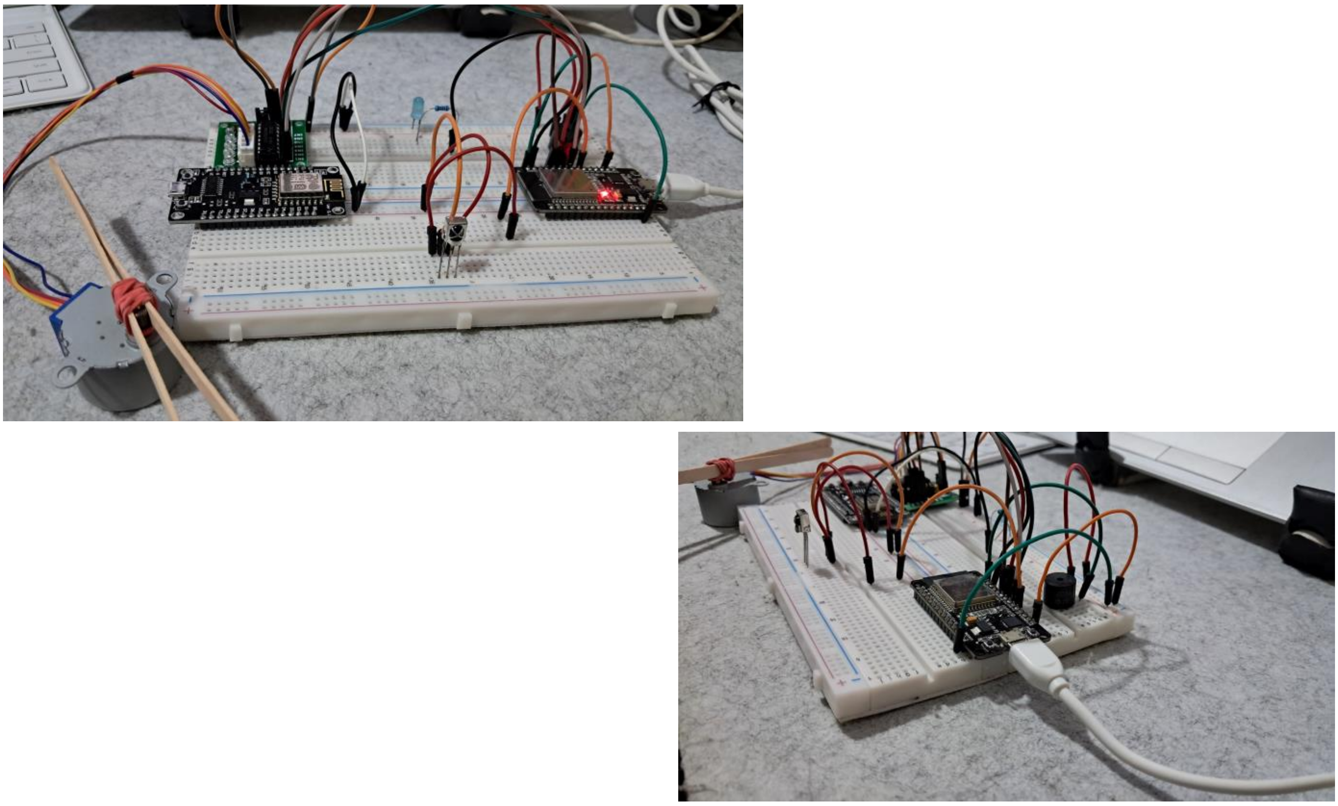

FIGURE 9: WORKING PROTOTYPE: ADDITIONAL PICTURES OF THE CIRCUIT

This is the final working prototype of the home appliance automation using IoT experiment, which successfully meets all the requirements.

CONCLUSION

In this experiment, I successfully built an automated appliance controller using the ESP32 dev module. I connected and programmed different components like a stepper motor (fan), led (light), buzzer (alarm), and IR sensor, all working together to simulate real home appliances. Using an IR remote, I could turn these appliances on or off wirelessly. I also used ThingSpeak to monitor the status of each appliance online. It felt like creating a simple version of a smart home system that can be controlled and tracked remotely.

Throughout the experiment, I learned how to set up both the hardware and the code properly. I had to fix wiring issues and make sure the code matched the actual connections. I also figured out how to use libraries like WiFi, ThingSpeak, and irRemote to connect to the internet and send data. Seeing the real-time graphs on ThingSpeak showing when each device was on or off was very satisfying.