دانشکده مهندسی کامپیوتر

درس برنامه‌نویسی تجهیزات اینترنت اشیا

پروژه پایانی

پیاده‌سازی دستیار هوشمند خانه با اتصال وای‌فای و کنترل از راه دور

با استفاده از ماژول ESP32-CAM

استاد درس: دکتر علی بهلولی

فاطمه صیادزاده، کیمیا کبیری

دی ماه ۱۴۰۳

گزارش پروژه کنترل وسایل باESP32 ، هوش مصنوعی و تلگرام

مقدمه

در این پروژه، یک سیستم هوشمند برای کنترل وسایل برقی از طریق اینترنت طراحی کردیم. در ابتدا، یک سرور محلی روی ESP32 راه‌اندازی شد که می‌توانست دستورات را از طریق یک فرم وب دریافت کند. سپس، سیستم گسترش یافت و قابلیت کنترل از طریق یک مدل هوش مصنوعی و ربات تلگرام نیز به آن اضافه شد. این سیستم قادر است با دریافت ورودی‌های متنی از کاربران، دستورات مناسب را تشخیص داده و به میکروکنترلر ارسال کند.

تجهیزات پروژه :

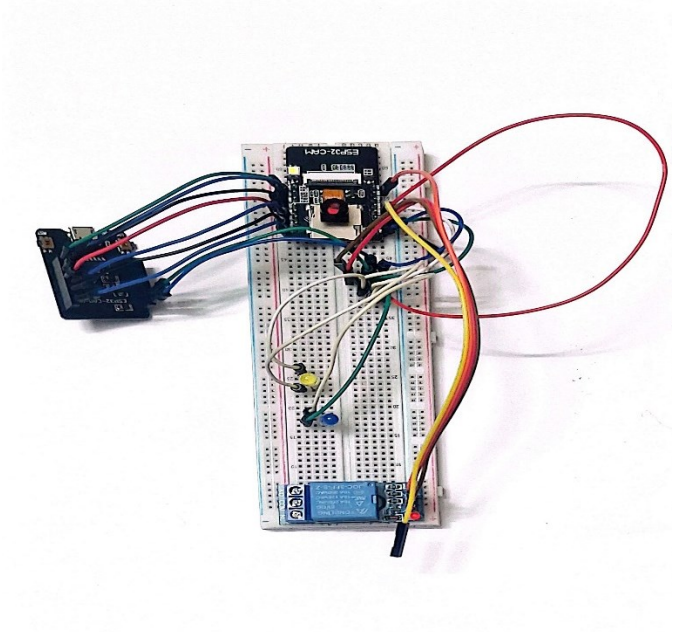
۱- برد بورد

۲- esp32-cam

۳- ۲ عدد led

۴- رله

۵- تعدادی سیم



---

۱. کد آردوینو (ESP32)

این بخش مسئول دریافت دستورات از طریق سرور وب و ارتباط سریال است. روش انتخابی ما Access Point بود.

۱/۱. تعریف و مقداردهی اولیه

```
#include <WiFi.h>
#include <WebServer.h>
```

ابتدا کتابخانه‌های مورد نیاز برای راه‌اندازی شبکه WiFi و سرور وب را اضافه می‌کنیم.

```
const char* ssid = "ESP32_AP";
const char* password = "12345678";
```

نام و رمز عبور نقطه دسترسی ESP32 (Access Point) را تنظیم می‌کنیم.

```
WebServer server(80);
#define KitchenLED 4
#define RoomLED 12
#define ParkingLED 13
```

متغییرها و پایه‌های مربوط به‌LED ها را تعریف میکنیم. همانطور که مشخص است پایه ٤، ۱۲ و ۱۳ به ترتیب برای آشپزخانه، اتاق و پارکینگ در نظر گرفته شده است.

۱/۲. صفحه HTML فرم وب

```
const char* formHTML = R"rawliteral(
<!DOCTYPE html>
<html>
<head>
 <title>ESP32 LED Control</title>
 <style>
  body {
   font-family: Arial, sans-serif;
   background:
url('https://png.pngtree.com/thumb_back/fh260/background/20230707/pngtree-d-
rendering-of-iot-devices-and-network-connection-in-the-internet-image_3766498.jpg')) no-
repeat center center fixed;
   background-size: cover;
   color: white;
   text-align: center;
   padding: 50px 0;
  }
  h2 {
   font-size: 2em;
```

```
    margin-bottom: 20px;
   }
   form {
    background-color: rgba(0, 0, 0, 0.6);
    padding: 30px;
    border-radius: 10px;
    display: inline-block;
   }
   label {
    font-size: 1.2em;
    margin-bottom: 10px;
   }
   input[type='text'] {
    padding: 10px;
    font-size: 1.2em;
    width: 300px;
    margin-bottom: 20px;
    border: none;
    border-radius: 5px;
    background-color: #f0f0f0;
   }
   input[type='submit'] {
    padding: 12px 30px;
    font-size: 1.2em;
    background-color: #4CAF50;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
   }
   input[type='submit']:hover {
    background-color: #45a049;
   }
  </style>
</head>
<body>
```

```
 <h2>ESP32 Data Form</h2>
 <form action="/submit" method="post">
  <label for="data">Enter data:</label><br><br>
  <input type="text" id="data" name="data" required><br><br>
  <input type="submit" value="Submit">
 </form>
</body>
</html>
)rawliteral";
```

در این بخش، یک فرم HTML برای ارسال داده‌ها به سرور طراحی شده است.

۱/۳. توابع کنترل وب سرور

```
void handleRoot() {
 server.send(200, "text/html", formHTML);
}
  digitalWrite(KitchenLED, LOW);
  delay(delayTime);
 }
}
```

در این بخش از کد، تابع handleRootوظیفه دارد که درخواست‌های ورودی به سرور وب داخلی ESP32 را مدیریت کرده و صفحه HTML مربوط به ورود داده را برای کاربر نمایش دهد. این صفحه از طریق مرورگر در دسترس قرار می‌گیرد و امکان ارسال داده به ESP32 را فراهم می‌کند. همچنین، برای کنترلLED ها، از دستور digitalWrite(KitchenLED, LOW); استفاده شده است که باعث خاموش شدن LED آشپزخانه می‌شود. تابع delay(delayTime); نیز تأخیری ایجاد می‌کند که می‌تواند برای چشمک زدن LED یا کنترل زمان‌بندی اجرای دستورات مفید باشد. این بخش از کد در کنار سایر بخش‌ها، امکان ارتباط تعاملی بین کاربر و دستگاه را فراهم می‌کند.

```
void handleSubmit() {
 if (server.method() == HTTP_POST) {
  if (server.hasArg("data")) {
   String data = server.arg("data");

   Serial.println(data);
```

```
    server.send(200, "text/html", "<h2>Data received: " + data + "</h2><a href='/'>Go
back</a>");
   } else {
    server.send(400, "text/plain", "Bad Request: Missing 'data'");
   }
  } else {
   server.send(405, "text/plain", "Method Not Allowed");
  }
}
```

تابع handleSubmit وظیفه دریافت داده‌های ارسال‌شده از طریق فرم HTML را بر عهده دارد. این تابع بررسی می‌کند که آیا درخواست دریافتی از نوع POST است یا نه. در صورتی که درخواست شامل مقدار data باشد، مقدار آن را دریافت کرده و در پورت سریال (Serial.println(data);) نمایش می‌دهد. همچنین، پاسخ HTML را برای کاربر ارسال می‌کند تا تأیید کند که داده با موفقیت دریافت شده است. در صورتی که مقدار dataدر درخواست وجود نداشته باشد، پیغام خطای Bad Requestارسال می‌شود. اگر نوع درخواست غیر از POSTباشد، سرور پاسخ Method Not Allowedرا برمی‌گرداند.

۱/٤. کنترل وضعیت LED ها

```
void setup() {
 Serial.begin(115200);
 pinMode(KitchenLED, OUTPUT);
 pinMode(RoomLED, OUTPUT);
 pinMode(ParkingLED, OUTPUT);

 digitalWrite(KitchenLED, LOW);
 digitalWrite(RoomLED, LOW);
 digitalWrite(ParkingLED, LOW);

 WiFi.softAP(ssid, password);
 Serial.println("Access Point started");
 Serial.print("IP address: ");
 Serial.println(WiFi.softAPIP());

 server.on("/", handleRoot);
 server.on("/submit", handleSubmit);

 server.begin();
```

```
 Serial.println("Web server started");
}
```

این بخش تنظیمات اولیه شامل مقداردهی پایه‌ها و راه‌اندازی سرور را انجام می‌دهد.
این تابع در هنگام راه‌اندازی ESP32 اجرا می‌شود و وظایف مقداردهی اولیه، تنظیمات شبکه و راه‌اندازی سرور وب را انجام می‌دهد. ابتدا ارتباط سریال با سرعت ۱۱۵۲۰۰ بیت بر ثانیه آغاز می‌شود تا داده‌های ورودی و خروجی در مانیتور سریال نمایش داده شوند. سپس پایه‌های مربوط به LED ها به‌عنوان خروجی تنظیم می‌شوند و در حالت پیش‌فرض خاموش می‌شوند. ESP32 در حالت نقطه دسترسی (Access Point) تنظیم می‌شود، بنابراین دستگاه‌های دیگر می‌توانند به آن متصل شوند. آدرس IP نقطه دسترسی در مانیتور سریال نمایش داده می‌شود. پس از آن، دو مسیر / و /submit برای دریافت و پردازش درخواست‌های کلاینت ثبت می‌شوند. در نهایت، سرور وب راه‌اندازی می‌شود تا درخواست‌های HTTP را مدیریت کند.

```
void loop() {
 server.handleClient();

 if (Serial.available()) {
  String data = Serial.readString();

  for (int i = 0; i < data.length(); i++) {
   char response = data.charAt(i);

   if (response == 'A') {
    digitalWrite(KitchenLED, HIGH);
    Serial.println("Kitchen LED turned ON.");
   }
   else if (response == 'B') {
    digitalWrite(KitchenLED, LOW);
    Serial.println("Kitchen LED turned OFF.");
   }
   else if (response == 'C') {
    digitalWrite(RoomLED, HIGH);
    Serial.println("Room LED turned ON.");
   }
   else if (response == 'D') {
    digitalWrite(RoomLED, LOW);
    Serial.println("Room LED turned OFF.");
   }
   else if (response == 'E') {
```

```
    digitalWrite(ParkingLED, HIGH);
    Serial.println("Parking LED turned ON.");
   }
   else if (response == 'F') {
    digitalWrite(ParkingLED, LOW);
    Serial.println("Parking LED turned OFF.");
   }
   else {
    Serial.println("⚠ Invalid command received.");
   }
  }
}
```

این تابع به‌صورت مداوم اجرا می‌شود و وظایف اصلی ESP32 را مدیریت می‌کند. ابتدا server.handleClient() اجرا می‌شود تا درخواست‌های دریافتی از طریق وب‌سرور پردازش شوند. سپس بررسی می‌شود که آیا داده‌ای از طریق پورت سریال دریافت شده است یا خیر. در صورت دریافت داده، حلقه‌ای اجرا شده و کاراکترهای دریافت‌شده پردازش می‌شوند.
برای هر کاراکتر دریافتی، دستورات مربوط به روشن یا خاموش کردن LED ها اجرا شده و نتیجه در مانیتور سریال نمایش داده می‌شود. اگر فرمان نامعتبری دریافت شود، هشدار خطا نمایش داده می‌شود. این تابع باعث می‌شود که ESP32 هم از طریق پورت سریال و هم از طریق وب‌سرور دستورات کنترل LED را دریافت و اجرا کند.

---

۲. کد پایتون (ارتباط سریال و هوش مصنوعی)

این بخش از کد پایتون وظیفه دریافت ورودی، ارسال آن به مدل هوش مصنوعی و انتقال دستور به ESP32 را دارد.

۲/۱ مقداردهی اولیه

```
import serial
import time
from langchain_openai import ChatOpenAI
```

اضافه کردن کتابخانه‌های مورد نیاز برای ارتباط سریال و مدل هوش مصنوعی.

```
last_request_time = 0
request_interval = 5
```

تعریف متغیرهایی برای جلوگیری از ارسال بیش از حد درخواست‌ها به هوش مصنوعی.

۲/۲ راه‌اندازی ارتباط سریال و هوش مصنوعی

```
def initialize_llm():
    """Initialize connection to AI model"""
    return ChatOpenAI(
        model="gpt-4o-mini",
        base_url="https://api.avalai.ir/v1",
        api_key="MyApiKey",
    )
```

این تابع اتصال به مدل هوش مصنوعی را برقرار می‌کند. از ChatOpenAI برای تنظیم مدل gpt-4o-mini استفاده شده است. همچنین، base_urlمشخص شده تا درخواست‌ها به API مربوطه ارسال شوند. در نهایت، api_key برای احراز هویت و دسترسی به سرویس هوش مصنوعی استفاده می‌شود. این تابع در هنگام راه‌اندازی برنامه اجرا شده و امکان برقراری ارتباط با مدل هوش مصنوعی را فراهم می‌کند.

```
def initialize_serial(port, baud_rate):
    """Initialize Serial communication with ESP32"""
    try:
        esp = serial.Serial(port, baud_rate, dsrdtr=False, rtscts=False, timeout=1)
        esp.dtr = False
        esp.rts = False
        time.sleep(1)  # Delay to ensure a stable connection
        return esp
    except serial.SerialException as e:
        print(f"❌ Error opening serial port: {e}")
        return None
```

این تابع برای مقداردهی اولیه ارتباط سریال بین کامپیوتر و ESP32 استفاده می‌شود. ابتدا با استفاده از serial.Serialیک اتصال سریال با نرخ انتقال داده‌ی مشخص (baud_rate) برقرار می‌شود. گزینه‌های dsrdtr=False و rtscts=False تنظیمات کنترلی برای جلوگیری از مشکلات ارتباطی هستند. سپس rts وdtr غیرفعال شده و با یک تأخیر کوتاه (۱ ثانیه) از پایداری ارتباط اطمینان حاصل می‌شود. در صورت موفقیت، شیء esp که به ارتباط سریال متصل است، برگردانده می‌شود؛ در غیر این صورت، پیام خطا در کنسول چاپ شده و مقدار None برگردانده می‌شود.

۲/۳ پردازش پیام‌ها

```
def process_user_input(llm, user_input):
    """Process user input and send it to AI"""
    global last_request_time
```

```
if time.time() - last_request_time < request_interval:
    print("⏳ Please wait a moment before sending another request.")
    return None

message = [
    {"role": "system", "content": """
    You are an AI assistant for an IoT system that controls LED lights.
    Based on the user's input, return a string of commands:
    A: Turn on kitchen LED
    B: Turn off kitchen LED
    C: Turn on room LED
    D: Turn off room LED
    E: Turn on parking LED
    F: Turn off parking LED
    Respond with a string like 'AC' to turn on both Kitchen and Room LEDs.
    """},
    {"role": "user", "content": user_input},
]

try:
    result = llm.invoke(message)
    commands = result.content.strip()
    # print(commands)
    valid_commands = ["A", "B", "C", "D", "E", "F"]
    if all(command in valid_commands for command in commands):
        last_request_time = time.time()  # Update last request time
        return commands
    else:
        print("❌ AI returned an invalid response. Please try again.")
except Exception as e:
    print(f"❌ Error communicating with AI: {e}")

return None
```

تابع process_user_input ابتدا بررسی می‌کند که آیا از آخرین درخواست حداقل ۵ ثانیه گذشته است تا از ارسال درخواست‌های مکرر و ایجاد فشار روی مدل هوش مصنوعی جلوگیری شود. در ادامه، یک پیام شامل دو بخش ایجاد می‌شود: بخش اول نقش سیستم را مشخص می‌کند که به مدل توضیح می‌دهد چگونه به‌عنوان یک دستیار برای کنترل

چراغ‌های LED عمل کند و فقط دستورات مشخص‌شده(A تا F) را در خروجی برگرداند، و بخش دوم که ورودی کاربر را برای پردازش به مدل ارسال می‌کند. سپس پیام از طریق تابع invokeبه مدل فرستاده شده و پاسخ آن دریافت می‌شود. پاسخ دریافتی پردازش شده و بررسی می‌شود که شامل فقط دستورات معتبر باشد. در صورت تأیید، مقدار آن بازگردانده و زمان آخرین درخواست به‌روز می‌شود، در غیر این صورت، پیام خطا نمایش داده خواهد شد. همچنین این تابع از قابلیت ارسال چندین دستور هم‌زمان پشتیبانی می‌کند، به این صورت که مدل می‌تواند ترکیبی از دستورات را در یک رشته ارائه دهد و چندین چراغ را به‌طور هم‌زمان کنترل کند.

---

۳. ارتقا کد و توسعه نسخه تلگرام

پس از پیاده‌سازی نسخه اولیه که از طریق ارتباط سریال با برد ESP32 کار می‌کرد، نسخه‌ای جدید با پشتیبانی از ربات تلگرام توسعه داده شد. این نسخه امکان ارسال دستورات از طریق تلگرام را فراهم می‌کرد و پیام‌های کاربران را پردازش و به ESP32 ارسال می‌نمود. با این حال، به دلیل مشکلات فیلترینگ و محدودیت‌های پروکسی، این نسخه در عمل ارتباط پایداری با سرورهای تلگرام نداشت و قابل استفاده نبود. حال به مقایسه این دو نسخه می‌پردازیم.

اضافه شدن ارتباط با تلگرام

در نسخه تلگرام، رباتی طراحی شد که پیام‌های کاربران را دریافت و به ESP32 ارسال می‌کرد. توابعی مانند handle_message برای پردازش پیام‌ها و دستورات /start و /help برای راهنمایی کاربران اضافه شدند.

حذف حلقه while

در نسخه اصلی، یک حلقه while اجرا می‌شد تا ورودی‌های ESP32 را بررسی کند، اما در نسخه تلگرام، این فرآیند به رویدادهای تلگرام واگذار شد و نیازی به اجرای دائمی حلقه نبود.

تغییر در نحوه دریافت ورودی

در نسخه اصلی، ورودی از طریق سریال دریافت می‌شد، اما در نسخه تلگرام، پیام‌های کاربران (update.message.text) مستقیماً پردازش می‌شدند.

تغییر در ارسال دستورات به ESP32

در نسخه اولیه، دستورات بر اساس داده‌های سریال ارسال می‌شدند. اما در نسخه تلگرام، پیام‌های کاربران ابتدا تحلیل و سپس از طریق سریال به ESP32 ارسال شدند.

جلوگیری از ارسال درخواست‌های مکرر

برای جلوگیری از فشار بیش از حد برESP32 ، در هر دو نسخه یک محدودیت زمانی ۵ ثانیه‌ای در نظر گرفته شد. در نسخه تلگرام، این موضوع اهمیت بیشتری داشت، زیرا کاربران ممکن بود چندین پیام متوالی ارسال کنند.

حذف flushInput() و flushOutput()

در نسخه اصلی، این دستورات برای مدیریت ارتباط سریالی استفاده می‌شدند، اما در نسخه تلگرام نیازی به آن‌ها نبود و حذف شدند.

۳/۱. راه‌اندازی ربات

```
from telegram.ext import Application, CommandHandler, MessageHandler, filters
```

اضافه کردن کتابخانه‌های مورد نیاز برای تلگرام.

```
def start_bot():
  token = "TOKEN"
  application = Application.builder().token(token).build()
  application.add_handler(CommandHandler("start", start))
  application.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, handle_message))
  application.run_polling()
```

در این تابع، توکن ربات تلگرام تعریف شده و پردازش پیام‌های دریافتی انجام می‌شود.

۳٫۲. ارسال دستورات به ESP32

```
async def handle_message(update, context):
  user_input = update.message.text
  command = process_user_input(initialize_llm(), user_input)
  if command:
    send_command_to_esp(command)
    await update.message.reply_text(f"✅ Command sent to ESP32: {command}")
```

این تابع پیام دریافتی از تلگرام را پردازش کرده و به ESP32 ارسال می‌کند.

نتیجه‌گیری

این پروژه با هدف طراحی یک سیستم کنترل از راه دور برای ESP32 با استفاده از فناوری‌های اینترنت اشیاء (IoT) پیاده‌سازی شد. در ابتدا، ارتباط سریالی میان کامپیوتر و ESP32 برقرار شد و سیستم قادر به ارسال دستورات به پلتفرم سخت‌افزاری برای کنترل دستگاه‌های مختلف مانندLED ها از طریق یک رابط کاربری ساده بود.

در مرحله بعد، نسخه‌ای با استفاده از ربات تلگرام توسعه داده شد تا کاربران بتوانند از طریق ارسال پیام‌های متنی دستورات خود را به ESP32 ارسال کنند. این نسخه به طور قابل توجهی تجربه کاربری را بهبود بخشید و امکان کنترل دستگاه‌ها را از هر مکانی به وسیله موبایل فراهم نمود.

با این حال، مشکلاتی مانند فیلترینگ و محدودیت‌های پروکسی در ارتباط با سرورهای تلگرام باعث شد که نسخه تلگرام نتواند به درستی عمل کند و با اختلال مواجه شود.

در نهایت، این پروژه نشان‌دهنده پتانسیل بالای استفاده از فناوری‌های مبتنی بر اینترنت اشیاء و ارتباطات تلگرامی برای ایجاد سیستم‌های هوشمند و کاربرپسند است. اگر مشکلات اتصال به تلگرام رفع شوند یا از فناوری‌های جایگزین برای ارتباط استفاده شود، می‌توان این سیستم را به یک ابزار مفید و کاربردی در محیط‌های مختلف تبدیل کرد.