

Instituto Superior de Tecnologias Avançadas do Porto

Ensino Superior

Projetos II - Laboratório de Hardware

Ano Letivo 2024/2025



Projeto de ESP32 - Invictaeria

José Couto, Rafael Baptista, José Oliveira, Tiago Carvalho

Porto, julho 2025

ISTEC

Índice

1. Introdução.....	3
1.1 Enquadramento.....	4
1.2 Objetivos do Projeto.....	4
2. ESP32.....	4
2.1 Características Principais.....	5
3. UML.....	6
4. Código Fonte.....	6
5. Resultados e Discussão.....	7
6. Projeto Físico.....	8
7. Conclusão.....	9

1. Introdução

O projeto *Invictaeria* consiste no desenvolvimento de uma solução IoT baseada em ESP32, com o objetivo de detetar e registar a passagem de aeronaves em tempo real. Este sistema capta sinais emitidos por aviões próximos, permitindo a sua monitorização através de dispositivos acessíveis e de baixo custo. A iniciativa alia hardware simples a técnicas de deteção e visualização de dados, promovendo uma maior consciência do tráfego aéreo em ambientes urbanos ou rurais. O projeto tem aplicações nas áreas da investigação, segurança, educação e cidadania ativa.

1.1 Enquadramento

O avanço das tecnologias IoT tem permitido o desenvolvimento de soluções acessíveis para monitorização do ambiente e recolha de dados em tempo real. Neste contexto, o projeto Invictaeria surge como uma aplicação prática focada na deteção de aeronaves que sobrevoam áreas específicas, utilizando para isso um microcontrolador ESP32. O sistema foi concebido no âmbito da unidade curricular Projetos II – Laboratório de Hardware, com o objetivo de explorar as capacidades de integração entre hardware e software num cenário do mundo real.

1.2 Objetivos do Projeto

O principal objetivo do projeto é desenvolver um protótipo funcional capaz de detetar sinais provenientes de aeronaves em voo e registar os dados obtidos para posterior visualização ou análise. Os objetivos específicos incluem:

- Implementar uma solução baseada em ESP32 com capacidade de captação de sinais relevantes;
- Garantir a recolha e armazenamento dos dados em tempo real;
- Explorar possíveis formas de visualização da informação recolhida;
- Avaliar a eficácia do sistema em diferentes ambientes de teste.

2. ESP32

O ESP32 é um microcontrolador de baixo custo e bom desempenho, com Wi-Fi e Bluetooth incluído. É muito usado em projetos de IoT porque é versátil, gasta pouca energia ,e tem muita documentação e suporte online. No projeto

Invictaeria, o ESP32 é o responsável por tratar da ligação à internet, receber os dados e mostrar as informações no ecrã OLED.

2.1 Características Principais

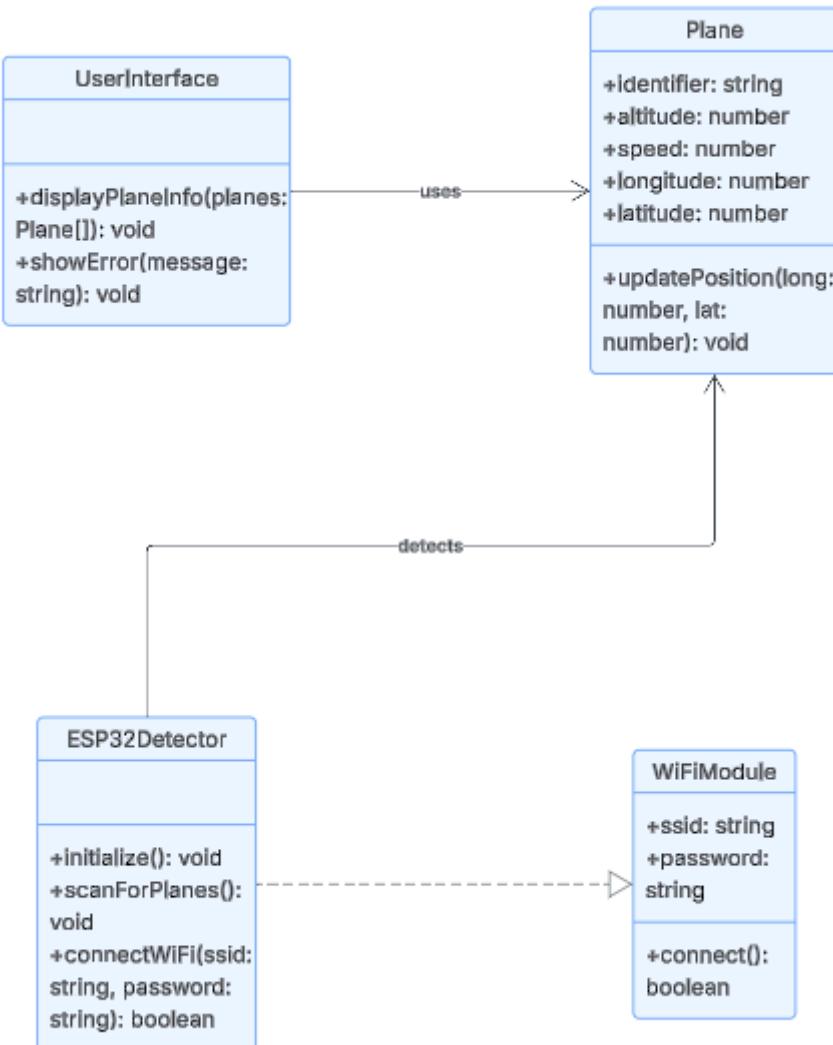
- Processador dual-core
- Wi-Fi 802.11 b/g/n
- Bluetooth BLE
- Suporte para várias interfaces (SPI, I2C, UART, etc)
- Compatível com Arduino IDE

Estas características tornam o ESP32 ideal para projetos pequenos mas com alguma complexidade como este.

Materiais utilizados

- **ESP32 C/bluetooth, wifi e módulo gps integrado ou não (Neo6M)**
- **Breadboard**
- **Jumpers**
- **Pequeno monitor (IC2)**
- **Buzzer**
- **LED**
- **Opcional/estético:**
 - **Placas de madeira para a caixa**
 - **Vidro**
 - **Antenas tp-link**
 - **Fita cola de dupla face**
 - **Cola para madeira**
 - **Lixa de água (para retirar as farpas da madeira)**
 - **Tinta preta**

3. UML

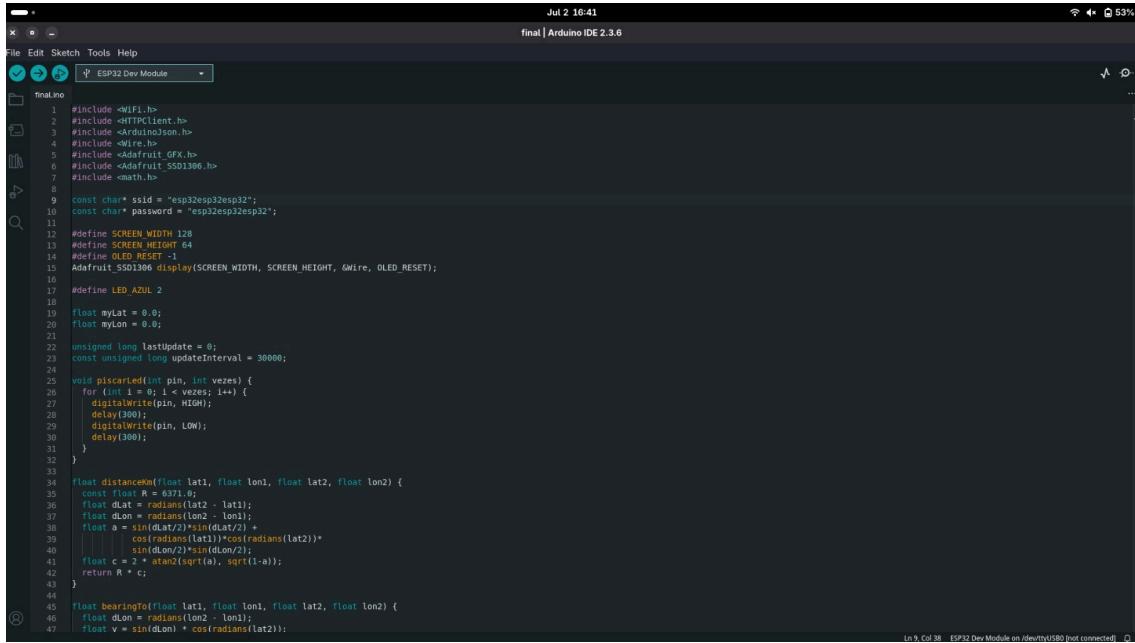


4. Código Fonte

O código foi feito no Arduino IDE, usando bibliotecas para o ESP32, o ecrã OLED e para fazer chamadas HTTP. Entre as partes mais importantes do código estão:

- Ligação à rede Wi-Fi a ser partilhada;
- Pedir dados da API OpenSky e tratar a resposta;
- Calcular a distância e direção com base nas coordenadas GPS;

- Mostrar tudo no ecrã de forma legível;
- Temporizador com contagem decrescente para saber quando vêm os próximos dados.



```

Jul 2 16:41
final | Arduino IDE 2.3.6

File Edit Sketch Tools Help
ESP32 Dev Module
final.ino
1 #include <WiFi.h>
2 #include <HTTPClient.h>
3 #include <arduinojson.h>
4 #include <Wire.h>
5 #include <Adafruit_GFX.h>
6 #include <Adafruit_SSD1306.h>
7 #include <math.h>
8
9 const char* ssid = "esp32esp32esp32";
10 const char* password = "esp32esp32esp32";
11
12 #define SCREEN_WIDTH 128
13 #define SCREEN_HEIGHT 64
14 #define OLED_RESET -1
15 Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
16
17 #define LED_AZUL 2
18
19 float myLat = 0.0;
20 float myLon = 0.0;
21
22 unsigned long lastUpdate = 0;
23 const unsigned long updateInterval = 30000;
24
25 void piscaLed(int pin, int vezes) {
26     for (int i = 0; i < vezes; i++) {
27         digitalWrite(pin, HIGH);
28         delay(100);
29         digitalWrite(pin, LOW);
30         delay(300);
31     }
32 }
33
34 float distanciaKm(float lat1, float lon1, float lat2, float lon2) {
35     const float R = 6371.0;
36     float dLat = radians(lat2 - lat1);
37     float dLon = radians(lon2 - lon1);
38     float a = sin(dLat/2) * sin(dLat/2) +
39               cos(radians(lat1)) * cos(radians(lat2)) *
40               sin(dLon/2) * sin(dLon/2);
41     float c = 2 * atan2(sqrt(a), sqrt(1-a));
42     return R * c;
43 }
44
45 float bearingTo(float lat1, float lon1, float lat2, float lon2) {
46     float dLon = radians(lon2 - lon1);
47     float v = sin(dLon) * cos(radians(lat2));

```

Line 9, Col 38 - ESP32 Dev Module on ideantly (USB0 front connected)

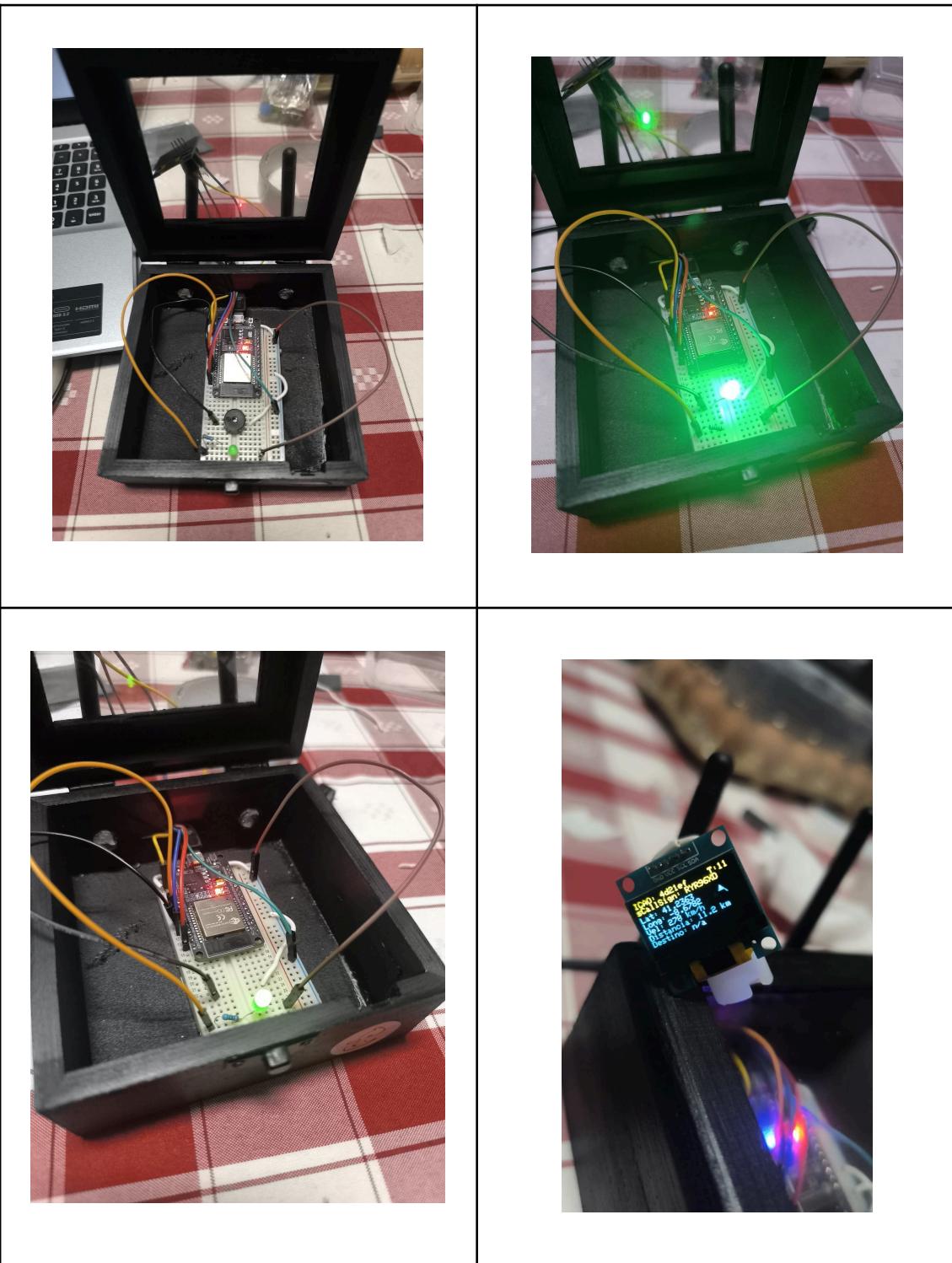
5. Resultados e Discussão

Os testes em simulação mostraram que o sistema funciona e consegue apanhar aviões por perto e mostrar a informação corretamente. Usar APIs públicas foi uma boa escolha porque simplificou muito o desenvolvimento. No entanto, há algumas coisas menos boas:

- O ecrã OLED é pequeno e não dá para mostrar tudo de uma vez, por isso, se adicionarmos mais alguma informação, precisamos de recorrer ao scroll.
- Para registar a partida e chegada do avião é necessário uma subscrição paga da API Open SKy ou FlightRadar24 (preferencial)

Estas limitações foram identificadas e podem ser melhoradas no futuro.

6. Projeto Físico



Código do programa

```
#include <WiFi.h>           // Liga o ESP32 a redes Wi-Fi
#include <HTTPClient.h>       // Permite fazer pedidos HTTP (GET, POST, etc.)
#include <ArduinoJson.h>       // Facilita o processamento de dados JSON
#include <Wire.h>             // Comunicação I2C com dispositivos (ex: OLED)
#include <Adafruit_GFX.h>       // Biblioteca base para gráficos (texto, formas)
#include <Adafruit_SSD1306.h>    // Controla ecrãs OLED SSD1306
#include <TinyGPS++.h>          // Processa dados do módulo GPS
#include <HardwareSerial.h>      // Permite usar UARTs adicionais no ESP32
#include <math.h>               // Funções matemáticas (ex: sin, cos, atan2)

// Configurações WiFi
const char* ssid = "esp32esp32esp32";
const char* password = "esp32esp32esp32";

// OLED Display
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// LED azul (status)
#define LED_AZUL 2

// GPS via UART2 (RX=16, TX=17)
TinyGPSPlus gps;
HardwareSerial gpsSerial(2);

// Variáveis para localização GPS
float myLat = 0.0;
float myLon = 0.0;

// Controle de atualização
unsigned long lastUpdate = 0;
const unsigned long updateInterval = 30000; // 30 segundos

// Pisca LED azul n vezes
void piscarLed(int pin, int vezes) {
    for (int i = 0; i < vezes; i++) {
        digitalWrite(pin, HIGH);
        delay(300);
        digitalWrite(pin, LOW);
```

```

        delay(300);
    }
}

// Calcula distância em km entre 2 pontos lat/lon usando fórmula haversine
float distanceKm(float lat1, float lon1, float lat2, float lon2) {
    const float R = 6371.0; // Raio da Terra em km
    float dLat = radians(lat2 - lat1);
    float dLon = radians(lon2 - lon1);
    float a = sin(dLat/2)*sin(dLat/2) +
        cos(radians(lat1))*cos(radians(lat2))*
        sin(dLon/2)*sin(dLon/2);
    float c = 2 * atan2(sqrt(a), sqrt(1-a));
    return R * c;
}

// Calcula rumo (bearing) de lat/lon1 até lat/lon2 em graus
float bearingTo(float lat1, float lon1, float lat2, float lon2) {
    float dLon = radians(lon2 - lon1);
    float y = sin(dLon) * cos(radians(lat2));
    float x = cos(radians(lat1)) * sin(radians(lat2)) -
        sin(radians(lat1)) * cos(radians(lat2)) * cos(dLon);
    return fmod((degrees(atan2(y, x)) + 360), 360);
}

// Desenha uma seta indicando direção na tela OLED
void drawHeadingArrow(int x, int y, float heading) {
    float rad = radians(heading - 90);
    int len = 10;
    int x2 = x + cos(rad) * len;
    int y2 = y + sin(rad) * len;
    display.drawLine(x, y, x2, y2, SSD1306_WHITE);
    display.fillTriangle(x2, y2, x2 - 3, y2 + 5, x2 + 3, y2 + 5, SSD1306_WHITE);
}

// Exibe barra de carregamento e conecta WiFi
void showLoadingBar() {
    int barWidth = 100, barHeight = 10;
    int barX = (SCREEN_WIDTH - barWidth) / 2;
    int barY = (SCREEN_HEIGHT - barHeight) / 2;
    const int totalSteps = 50;

    WiFi.begin(ssid, password);

    for (int currentStep = 0; currentStep <= totalSteps; currentStep++) {
        display.clearDisplay();
}

```

```
display.setTextSize(1);
display.setCursor((SCREEN_WIDTH - 80) / 2, barY - 15);
display.print("A ligar WiFi... ");
display.drawRect(barX, barY, barWidth, barHeight, SSD1306_WHITE);
int fillWidth = (barWidth - 2) * currentStep / totalSteps;
if (fillWidth > 0) {
    display.fillRect(barX + 1, barY + 1, fillWidth, barHeight - 2, SSD1306_WHITE);
}
display.display();
delay(100);

if (WiFi.status() == WL_CONNECTED) break;
}

if (WiFi.status() == WL_CONNECTED) {
    display.clearDisplay();
    display.setTextSize(1);
    display.setCursor((SCREEN_WIDTH - 80) / 2, barY - 15);
    display.print("WiFi ligado!");
    display.drawRect(barX, barY, barWidth, barHeight, SSD1306_WHITE);
    display.fillRect(barX + 1, barY + 1, barWidth - 2, barHeight - 2, SSD1306_WHITE);
    display.display();
    delay(500);
    piscarLed(LED_AZUL, 3);
    digitalWrite(LED_AZUL, HIGH);
} else {
    display.clearDisplay();
    display.setCursor(0, 0);
    display.setTextSize(1);
    display.println("Falha ao conectar WiFi.");
    display.display();
    while (true) delay(1000);
}
}

// Lê dados do GPS por até 5 segundos e atualiza myLat/myLon
bool readGPSLocation() {
    unsigned long start = millis();
    while (millis() - start < 5000) {
        while (gpsSerial.available() > 0) {
            if (gps.encode(gpsSerial.read())) {
                if (gps.location.isValid()) {
                    myLat = gps.location.lat();
                    myLon = gps.location.lng();
                    return true;
                }
            }
        }
    }
}
```

```

        }
    }
}

return false; // Falha em obter localização válida
}

// Busca dados dos aviões próximos via OpenSky API e exibe no OLED
bool getPlaneData() {
    if (WiFi.status() != WL_CONNECTED) return false;

    HttpClient http;
    float lamin = myLat - 0.5;
    float lamax = myLat + 0.5;
    float lomin = myLon - 0.5;
    float lomax = myLon + 0.5;

    String url = "https://opensky-network.org/api/states/all?lamin=" + String(lamin, 6) +
        "&lamax=" + String(lamax, 6) +
        "&lomin=" + String(lomin, 6) +
        "&lomax=" + String(lomax, 6);

    http.begin(url);
    int httpCode = http.GET();

    if (httpCode == 200) {
        String payload = http.getString();
        DynamicJsonDocument doc(5000);
        DeserializationError error = deserializeJson(doc, payload);
        if (error) {
            http.end();
            return false;
        }

        JSONArray states = doc["states"];
        if (states.size() == 0) {
            http.end();
            return false;
        }

        float closestDist = 99999.0;
        JSONArray closest;

        for (JsonVariant v : states) {
            JSONArray a = v.as<JSONArray>();
            if (!a[6].isNull() && !a[5].isNull()) {
                float dist = distanceKm(myLat, myLon, a[6], a[5]);
                if (dist < closestDist) {
                    closestDist = dist;
                    closest = a;
                }
            }
        }

        if (closest != null) {
            String closestString = closest.toString();
            Serial.print("Closest plane: ");
            Serial.println(closestString);
        }
    }
}

```

```

if (dist < closestDist) {
    closestDist = dist;
    closest = a;
}
}

if (closest.size() > 0) {
    display.clearDisplay();
    display.setTextSize(1);

    int y = 0;
    display.setCursor(0, y);
    display.print("ICAO: ");
    display.println(closest[0].as<const char*>());

    y += 8;
    display.setCursor(0, y);
    display.print("CallSign: ");
    display.println(closest[1].as<const char*>());

    y += 8;
    display.setCursor(0, y);
    display.printf("Lat: %.4f", closest[6].as<float>());

    y += 8;
    display.setCursor(0, y);
    display.printf("Long: %.4f", closest[5].as<float>());

    y += 8;
    display.setCursor(0, y);
    display.printf("Vel: %.0f km/h", closest[9].as<float>() * 3.6);

    y += 8;
    display.setCursor(0, y);
    display.printf("Distancia: %.1f km", closestDist);

    y += 8;
    display.setCursor(0, y);
    display.print("Origem: ");
    display.println(closest[2].as<const char*>());

// Desenha seta apontando para o avião (posição fixa no OLED)
float heading = bearingTo(myLat, myLon, closest[6], closest[5]);
drawHeadingArrow(110, 30, heading);

```

```
    display.display();
}

http.end();
return true;
}

http.end();
return false;
}

// Mostra temporizador regressivo no canto superior direito
void showTimer() {
    int secondsLeft = (updateInterval - (millis() - lastUpdate)) / 1000;
    display.setTextSize(1);
    display.setTextColor(SSD1306_WHITE, SSD1306_BLACK); // sobrescreve fundo
    display.setCursor(SCREEN_WIDTH - 28, 0);
    display.printf("T:%02ds", secondsLeft);
    display.display();
}

void setup() {
    Serial.begin(115200);
    pinMode(LED_AZUL, OUTPUT);
    digitalWrite(LED_AZUL, LOW);

    // Inicializa OLED
    if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
        Serial.println("Falha OLED");
        while (true);
    }

    // Mostra barra de conexão WiFi
    showLoadingBar();

    // Tela inicial
    display.clearDisplay();
    display.setTextSize(2);
    display.setCursor((SCREEN_WIDTH - 12 * 9) / 2, 20);
    display.println("Invictaeria");
    display.display();
    delay(1500);

    // Inicializa UART2 para GPS: RX=16, TX=17, baud 9600 (NEO-6M DEFAULT)
    gpsSerial.begin(9600, SERIAL_8N1, 16, 17);
```

```
// Tenta pegar localização GPS válida
if (!readGPSLocation()) {
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println("GPS sem sinal");
    display.display();
    while (true);
}

// Atualiza dados dos aviões pela primeira vez
getPlaneData();
lastUpdate = millis();
}

void loop() {
    unsigned long now = millis();

    // Atualiza dados dos aviões a cada 30 segundos
    if (now - lastUpdate >= updateInterval) {
        lastUpdate = now;
        getPlaneData();
    }

    // Mostra temporizador regressivo no canto da tela
    showTimer();

    delay(1000);
}
```

Dificuldades

Ao longo do projeto, uma das coisas que mais nos desorientou foi perceber como é que o GPS funciona com o ESP32. Não estavamos à espera de ser tão chato de interpretar os dados que ele manda, e fazer com que desse coordenadas certas levou algum tempo e vários testes. A parte da seta no ecrã também deu trabalho, porque tivemos de transformar o ângulo numa direção que fizesse sentido visualmente, o que não é tão direto como parece. Mesmo assim, ver aquilo a funcionar no fim deu-nos gosto. Se fosse possível usar uma antena para apanhar os sinais dos aviões em vez de depender de APIs, acho que o projeto ficava ainda mais interessante.

7. Conclusão

O projeto Invictaeria mostrou que é possível fazer um sistema de deteção de aviões simples e útil com poucos recursos. O ESP32, o ecrã OLED e o uso de APIs permitiram criar um protótipo funcional, que pode ser usado em contexto educativo ou mesmo para curiosidade pessoal. Para o futuro, dava jeito adicionar por exemplo envio dos dados para uma base de dados online, ou guardar os dados localmente com um cartão SD.