

# **SEL0614 - Aplicação de Microprocessadores**

Projeto Final: Controlador de Voo para Aeronave com RTOS

Pedro Porto Teixeira - Nº USP: 14603436

Renan Correia Monteiro Soares - Nº USP: 14605661

Vitor Alexandre Garcia Vaz - Nº USP: 14611432

10 de dezembro de 2025



**EESC - USP<sup>®</sup>**

# Sumário

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Ambiente de Desenvolvimento e PlatformIO</b>	<b>3</b>
2.1	Configuração do Projeto . . . . .	3
2.2	Gerenciamento de Dependências e Versões . . . . .	3
<b>3</b>	<b>Movimentação da Aeronave e Implementação de Controle</b>	<b>4</b>
3.1	Arfagem (Pitch) - Profundor . . . . .	4
3.2	Alternância entre Rolagem e Guinada . . . . .	5
3.3	Rolagem (Roll) - Ailerons . . . . .	5
3.4	Guinada (Yaw) - Leme . . . . .	5
<b>4</b>	<b>Mapeamento de Hardware</b>	<b>5</b>
4.1	Considerações sobre a Pinagem . . . . .	6
4.2	Visualização do circuito . . . . .	7
<b>5</b>	<b>Arquitetura do Sistema e FreeRTOS</b>	<b>7</b>
5.1	Tabela de Tasks . . . . .	7
5.2	Mecanismos de Comunicação e Sincronização . . . . .	8
5.2.1	Queues (Filas) . . . . .	8
5.2.2	Software Timer . . . . .	8
5.2.3	Interrupção Externa (ISR) . . . . .	8
<b>6</b>	<b>Implementação e Código</b>	<b>8</b>
6.1	Definição das Estruturas de Dados . . . . .	8
6.2	Task de Leitura (Joystick) . . . . .	9
6.3	Task de Controle (Atuação) . . . . .	9
<b>7</b>	<b>Conclusão</b>	<b>10</b>

# 1 Introdução

Este relatório apresenta o desenvolvimento de um sistema embarcado para controle de superfícies móveis de uma aeronave (ailerons, flaps, leme e profundor), conforme proposto na opção 2 do Projeto Final da disciplina SEL0614.

O sistema foi desenvolvido utilizando o microcontrolador ESP32 e o sistema operacional de tempo real **FreeRTOS**. O projeto integra a leitura de um joystick analógico para comando, servos motores para atuação nas superfícies, um display LCD para feedback visual do estado de voo e um cartão SD para registro (datalogging) das operações.

## 2 Ambiente de Desenvolvimento e PlatformIO

Para o desenvolvimento do firmware, optou-se pela utilização do **PlatformIO**, um ecossistema profissional para desenvolvimento de sistemas embarcados integrado ao Visual Studio Code. A escolha desta ferramenta em detrimento da IDE padrão do Arduino deve-se à sua capacidade superior de gerenciamento de dependências, controle de versões de bibliotecas e facilidade de compilação.

A configuração completa do ambiente está centralizada no arquivo `platformio.ini`, que define as regras de compilação e as ferramentas necessárias para o microcontrolador ESP32.

### 2.1 Configuração do Projeto

O arquivo de configuração utilizado no projeto é apresentado abaixo. Ele instrui o sistema a utilizar o framework Arduino sobre a plataforma da Espressif.

```
1 [env:esp32dev]
2 platform = espressif32
3 board = esp32dev
4 framework = arduino
5 platform_packages =
6     framework-arduinoespressif32 @ https://github.com/espressif/arduino-
7         esp32.git#3.0.2
8     framework-arduinoespressif32-libs @ https://github.com/espressif/
9         arduino-esp32/releases/download/3.0.2/esp32-arduino-libs-3.0.2.zip
8 lib_deps = marcoschwartz/LiquidCrystal_I2C@~1.1.4
9 monitor_speed = 115200
```

Listing 1: Arquivo de Configuração `platformio.ini`

### 2.2 Gerenciamento de Dependências e Versões

A utilização do PlatformIO trouxe benefícios cruciais para a organização do projeto, conforme detalhado nos parâmetros do arquivo de configuração:

- **Gerenciamento de Bibliotecas (`lib_deps`):** A biblioteca `LiquidCrystal_I2C` (versão 1.1.4), necessária para o funcionamento do display LCD, é declarada explicitamente. O PlatformIO encarrega-se de baixar e instalar essa biblioteca automaticamente em qualquer computador onde o projeto for aberto, eliminando a necessidade de instalação manual de drivers ou bibliotecas externas.

- **Versionamento do Core (platform\_packages):** O projeto fixa uma versão específica do core do Arduino para ESP32 (versão 3.0.2), baixada diretamente do repositório oficial. Isso garante que atualizações futuras da plataforma não quebrem a compatibilidade do código (especialmente em relação às funções do FreeRTOS e timers), assegurando a reproduzibilidade do firmware.
- **Monitor Serial (monitor\_speed):** A velocidade de comunicação serial foi padronizada em 115200 bps para depuração (debug) via console, alinhada com a inicialização da Serial no código principal (`Serial.begin(115200)`).

Dessa forma, o arquivo `platformio.ini` atua como um "container" de configuração, garantindo que o ambiente de compilação seja idêntico para todos os desenvolvedores envolvidos no projeto.

### 3 Movimentação da Aeronave e Implementação de Controle

A lógica de movimentação da aeronave foi projetada para permitir o controle dos três eixos principais de rotação — arfagem (pitch), rolagem (roll) e guinada (yaw) — utilizando um único joystick analógico de dois eixos (X e Y). Para contornar a limitação física do joystick, foi implementado um sistema de alternância de modos de controle via botão.

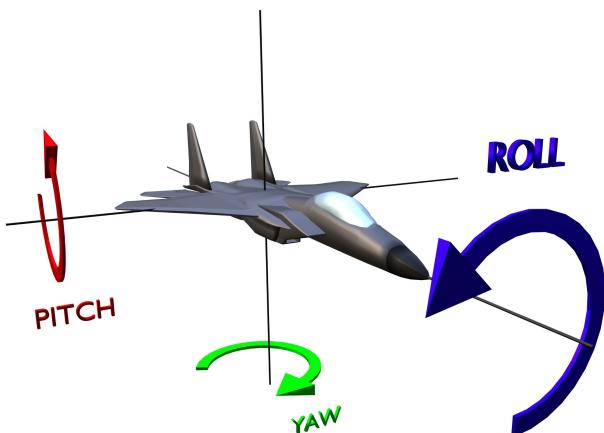


Figura 1: Eixos de movimentação controlados: Pitch (Arfagem), Roll (Rolagem) eYaw (Guinada).

#### 3.1 Arfagem (Pitch) - Profundor

O controle de arfagem, responsável por fazer o avião subir ou descer, está permanentemente associado ao eixo Y do joystick. A leitura analógica desse eixo é convertida em uma variação angular que atua sobre os servos do profundor (Elevator).

No código, ambos os servos do profundor movem-se na mesma direção para alterar a inclinação da aeronave:

```

1 // Calcula variações de ângulo baseadas nas leituras do joystick
2 int16_t angleVariationY = analogReadToAngleVariation(receivedControlData
    .yValue);
3
4 // Ajusta os servos dos profundores (movimento sincronizado)
5 ElevatorLeftServo.setAngle(90 + angleVariationY);
6 ElevatorRightServo.setAngle(90 + angleVariationY);

```

Listing 2: Controle do Profundor (Pitch)

### 3.2 Alternância entre Rolagem e Guinada

Como o eixo X do joystick precisa controlar tanto a rolagem (Ailerons) quanto a guinada (Leme), foi utilizada a interrupção externa do botão do joystick (pino 27) para alternar entre estes dois modos.

A função de callback da interrupção (`vIsrYawToRollChangeCallback`) envia um sinal para a task de controle, alterando a variável de estado `xMoviemnt`.

### 3.3 Rolagem (Roll) - Ailerons

Quando o sistema está no modo `ROLL_MOVIEMENT`, o movimento lateral do joystick atua sobre os ailerons. Para realizar a rolagem, os ailerons devem mover-se de forma diferencial (um sobe e o outro desce).

A implementação reflete essa física invertendo o sinal da variação angular para um dos servos:

```

1 if(receivedControlData.xMoviemnt == ROLL_MOVIEMENT){
2     // Ailerons movem-se em direções opostas para criar torque de rolagem
3     AileronLeftServo.setAngle(90 - angleVariationX);
4     AileronRightServo.setAngle(90 + angleVariationX);
5 }

```

Listing 3: Controle dos Ailerons (Roll)

### 3.4 Guinada (Yaw) - Leme

No modo `YAW_MOVIEMENT`, o eixo X passa a controlar o Leme (Rudder). Este movimento permite que a aeronave realize curvas no plano horizontal ("glissar"). A atuação é direta sobre o servo do leme:

```

1 if(receivedControlData.xMoviemnt == YAW_MOVIEMENT){
2     // Atuação direta no leme vertical
3     RudderServo.setAngle(90 + angleVariationX);
4 }

```

Listing 4: Controle do Leme (Yaw)

## 4 Mapeamento de Hardware

A conexão dos periféricos ao microcontrolador ESP32 segue as definições estabelecidas no firmware. A Tabela 1 detalha a atribuição de pinos (GPIOs) para cada componente do sistema, incluindo interfaces de comunicação e canais PWM.

Tabela 1: Tabela de Pinagem do Sistema (ESP32)

Componente / Função	GPIO	Observações
<b>Interface de Usuário (Display)</b>		
Display SDA	21	Comunicação I2C
Display SCL	22	Comunicação I2C
<b>Armazenamento (Cartão SD)</b>		
SD Chip Select (CS)	15	Controle SPI
SD SCK	14	Clock SPI (Padrão ESP32)
SD MISO	12	Dados SPI (Padrão ESP32)
SD MOSI	13	Dados SPI (Padrão ESP32)
<b>Atuadores (Servomotores)</b>		
Servo Aileron Esquerdo	26	Canal PWM 2
Servo Aileron Direito	25	Canal PWM 3
Servo Profundor Esquerdo	18	Canal PWM 4
Servo Profundor Direito	19	Canal PWM 5
Servo Leme	5	Canal PWM 6
<b>Controles de Entrada (Joystick)</b>		
Eixo X	34	Entrada Analógica (ADC1)
Eixo Y	35	Entrada Analógica (ADC1)
Botão de Seleção	27	Input Pull-up / Interrupção

## 4.1 Considerações sobre a Pinagem

- **Entradas Analógicas:** Os pinos 34 e 35 foram escolhidos para o joystick pois são pinos exclusivos de entrada (GPI) no ESP32, ideais para leitura de sensores analógicos de alta impedância.
- **SPI do Cartão SD:** Embora apenas o pino CS (15) esteja explicitamente definido via macro no código principal, a biblioteca `SDCardLogger` utiliza a interface SPI padrão do hardware ESP32 (SCK=14, MISO=12, MOSI=13) para comunicação de dados.
- **Botão:** O pino 27 utiliza o resistor de *pull-up* interno (`INPUT_PULLUP`) e está configurado para disparar uma interrupção na borda de descida (`FALLING`), detectando quando o usuário pressiona o botão para alternar o modo de controle.

## 4.2 Visualização do circuito

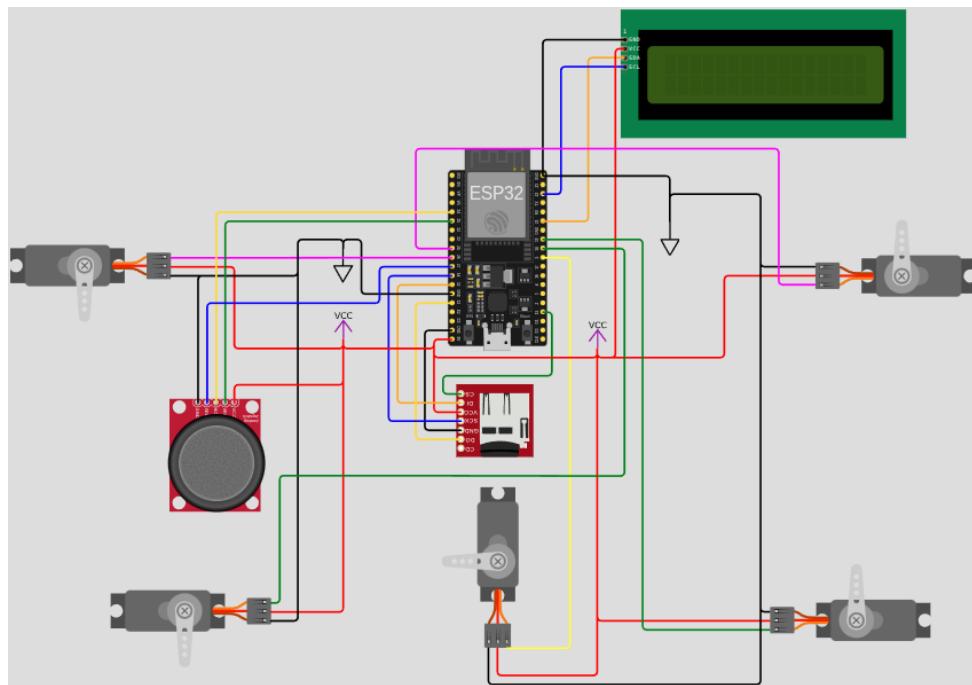


Figura 2: Imagem ilustrativa do circuito.

## 5 Arquitetura do Sistema e FreeRTOS

A robustez do sistema é garantida pela divisão das responsabilidades em tarefas (Tasks) independentes gerenciadas pelo FreeRTOS, permitindo o uso eficiente dos dois núcleos (Dual Core) do ESP32.

### 5.1 Tabela de Tasks

A Tabela 2 descreve as tarefas criadas, suas prioridades, o núcleo de processamento (Core) atribuído e suas responsabilidades.

Tabela 2: Definição das Tasks do Sistema

Task Handle	Prio.	Core	Descrição
vDisplayWriteTask	2	0 (PRO)	Responsável por receber o estado atual de voo e atualizar o display LCD 16x2.
vSDCardSaveTask	2	1 (APP)	Recebe dados de voo e realiza a gravação (log) no cartão SD com timestamp.
vJoystickReadTask	1	1 (APP)	Lê os valores ADCs do joystick, interpreta a lógica de voo e distribui os dados.
vControlTask	1	0 (PRO)	Recebe os comandos e aciona os servos motores (Leme, Ailerons, Profundor).

## 5.2 Mecanismos de Comunicação e Sincronização

Para garantir a integridade dos dados e o desacoplamento entre as tarefas, foram utilizados os seguintes recursos do FreeRTOS:

### 5.2.1 Queues (Filas)

As filas foram utilizadas para passar mensagens entre a tarefa produtora de dados (leitura do joystick) e as tarefas consumidoras (Display, SD e Controle).

- `xFlightStateToDisplay`, `xFlightStateToSDCard`, `xFlightStateToControl`: Distribuem o estado de voo calculado (enum `flightState`) para os periféricos.
- `xChangeYawOrRollToControl`: Fila específica para comunicar a mudança de modo de controle (Leme ou Aileron) vinda da interrupção.

### 5.2.2 Software Timer

Foi utilizado um *Software Timer* (`xJoystickReadTimerHandle`) configurado com período de 70ms. Ao invés de usar `delay()` bloqueante dentro da task de leitura, o timer dispara um callback que notifica a task `vJoystickReadTask`. Isso garante uma amostragem precisa e periódica das entradas analógicas.

### 5.2.3 Interrupção Externa (ISR)

O botão do joystick (Pino 27) está configurado como uma interrupção externa. A função `vIsrYawToRollChangeCallback` é acionada na borda de descida (FALLING).

- **Função:** Alternar o controle do eixo X do joystick entre guinada (*Yaw/Leme*) e rolagem (*Roll/Aileron*).
- A ISR envia o novo estado para uma fila (`xChangeYawOrRollToControl`) usando a função segura `xQueueSendFromISR`, evitando processamento pesado dentro da rotina de interrupção.

## 6 Implementação e Código

Abaixo são apresentados e explicados os trechos fundamentais do código desenvolvido.

### 6.1 Definição das Estruturas de Dados

Para organizar a comunicação entre as tasks, foram criadas estruturas e enums que representam o estado da aeronave.

```
1 // Enum de possíveis estados de voo
2 typedef enum{
3     CRUISE = 0U,
4     PITCH_UP ,
5     PITCH_DOWN ,
6     YAW_RIGHT , // ... outros estados
7 } flightState;
8
9 // Dado de controle completo
```

```

10 typedef struct{
11     flightState state;
12     xMoviemntState xMoviemnt; // Define se X controla Yaw ou Roll
13     uint16_t xValue;
14     uint16_t yValue;
15 } controlData;

```

Listing 5: Estruturas de Dados e Estados

**Explicação:** A estrutura `controlData` encapsula todas as informações necessárias para que a Task de Controle saiba exatamente como mover os servos, sem precisar acessar variáveis globais ou ler o hardware diretamente.

## 6.2 Task de Leitura (Joystick)

Esta é a task central que processa a entrada e distribui as decisões.

```

1 void vJoystickReadTask(void *pvParameters){
2     xTimerStart(xJoystickReadTimerHandle, 0); // Inicia o timer periodico
3
4     while(1){
5         // Espera pela notificacao do timer (a cada 70ms)
6         ulTaskNotifyTake(pdTRUE, portMAX_DELAY);
7
8         // Verifica se houve troca de modo (Botao pressionado)
9         if(xQueueReceive(xChangeYawOrRollToControl, &currentXMoviemntState,
10            0) == pdTRUE){
11             controlData.xMoviemnt = currentXMoviemntState;
12         }
13
14         // Leitura ADC e Logica de Estado
15         controlData.xValue = analogRead(JOYSTICK_ADC_X_PIN);
16         controlData.yValue = analogRead(JOYSTICK_ADC_y_PIN);
17
18         if(controlData.yValue > 3200) currentFlightState = PITCH_UP;
19         else if(controlData.yValue < 2400) currentFlightState = PITCH_DOWN;
20         // ... logica para Roll/Yaw
21
22         // Envio para as filas de saida
23         xQueueSend(xFlightStateToDisplay, &currentFlightState, 0);
24         xQueueSend(xFlightStateToSDCard, &currentFlightState, 0);
25         xQueueSend(xFlightStateToControl, &controlData, 0);
26     }
}

```

Listing 6: Lógica da Task de Leitura

**Explicação:** A task permanece bloqueada até receber a notificação do timer. Ao acordar, ela verifica se o usuário apertou o botão (via fila da ISR), lê os ADCs, determina se o avião deve subir, descer ou virar, e envia esses dados para três filas distintas simultaneamente.

## 6.3 Task de Controle (Atuação)

Esta task consome os dados processados e move os servos fisicamente.

```

1 void vControlTask(void *pvParameters){
2     controlData receivedControlData;

```

```

3 controlData.oldControlData = {CRUISE, YAW_MOVIMENT, 2048, 2048};
4
5 while(1){
6     // Bloqueia ate receber novo comando
7     xQueueReceive(xFlightStateToControl, &receivedControlData,
8     portMAX_DELAY);
9
10    // Filtragem: se o dado for igual ao anterior, nao faz nada (
11    // economiza CPU/Servo)
12    if( /* comparacao com oldControlData */ ) continue;
13
14    // Logica de mixagem e atuacao
15    int16_t angleVariationX = analogReadToAngleVariation(
16        receivedControlData.xValue);
17    int16_t angleVariationY = analogReadToAngleVariation(
18        receivedControlData.yValue);
19
20    // Profundor sempre atua com eixo Y
21    ElevatorLeftServo.setAngle(90 + angleVariationY);
22    ElevatorRightServo.setAngle(90 + angleVariationY);
23
24    // Eixo X atua no Leme OU Aileron dependendo do modo selecionado
25    if(receivedControlData.xMoviemt == YAW_MOVIMENT){
26        RudderServo.setAngle(90 + angleVariationX);
27    } else {
28        // Ailerons movem-se em direcoes opostas
29        AileronLeftServo.setAngle(90 - angleVariationX);
30        AileronRightServo.setAngle(90 + angleVariationX);
31    }
32}
33}

```

Listing 7: Controle dos Servomotores

**Explicação:** A lógica de controle implementa diferenciação para os ailerons (um sobe, outro desce para realizar a rolagem) e atuação direta para profundor e leme. O uso de uma verificação de estado anterior (`oldControlData`) evita "jitters" (tremores) nos servos se o joystick estiver parado.

## 7 Conclusão

O projeto atendeu aos requisitos propostos, implementando um controlador de voo funcional capaz de gerenciar múltiplas superfícies de controle simultaneamente. O uso do FreeRTOS foi essencial para garantir que a escrita no cartão SD (que pode ser lenta) não interferisse na resposta imediata dos servos ao comando do joystick, demonstrando a eficácia de sistemas operacionais de tempo real em aplicações aviônicas críticas.