

# Projeto Sistemas Embarcados

## Projeto Sistemas Embarcados

**Etapa 3 - Biblioteca para ler os ângulos Roll e Pitch gerados por um IMU e controlar os ângulos de 2 servomotores.**

**Membros:** Antônio Farias Araújo Terceiro, Joao Marcos Amorim de Almeida, Jorge Vinícius Santos Castro, Pedro Macêdo Luna, Rennyson Cavalcante Soares.

Ferramentas necessárias:

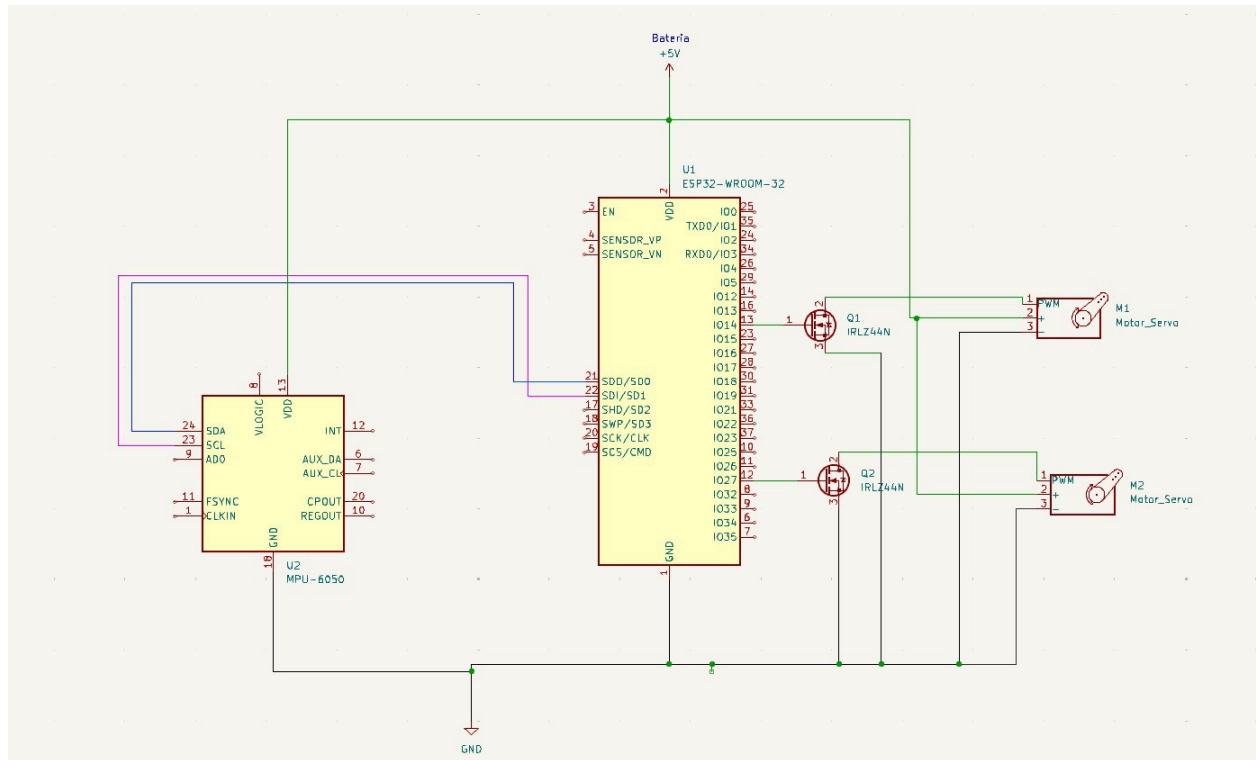
- Visual Studio Code: [Download](#)
- ESP-IDF: [Download](#)

Hardware Necessário:

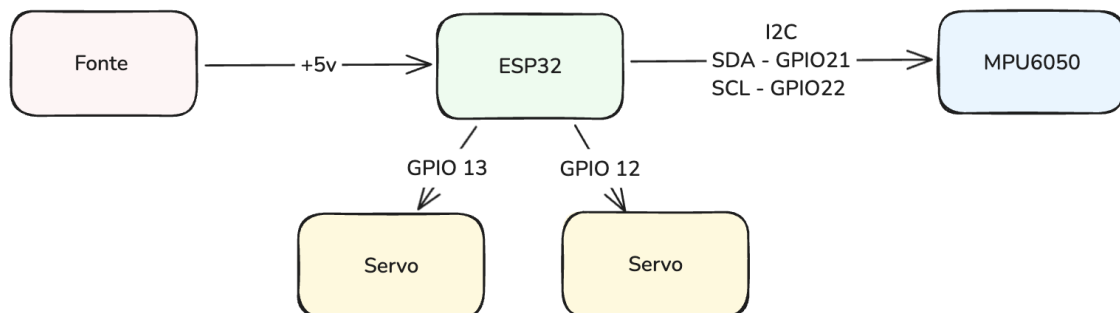
- ESP-32-WROOM
- MPU-6050:
  - [Documentação](#)
  - [Wokwi Component](#)
- 2 SERVOS:
  - [Wokwi Component](#)

# Firmware

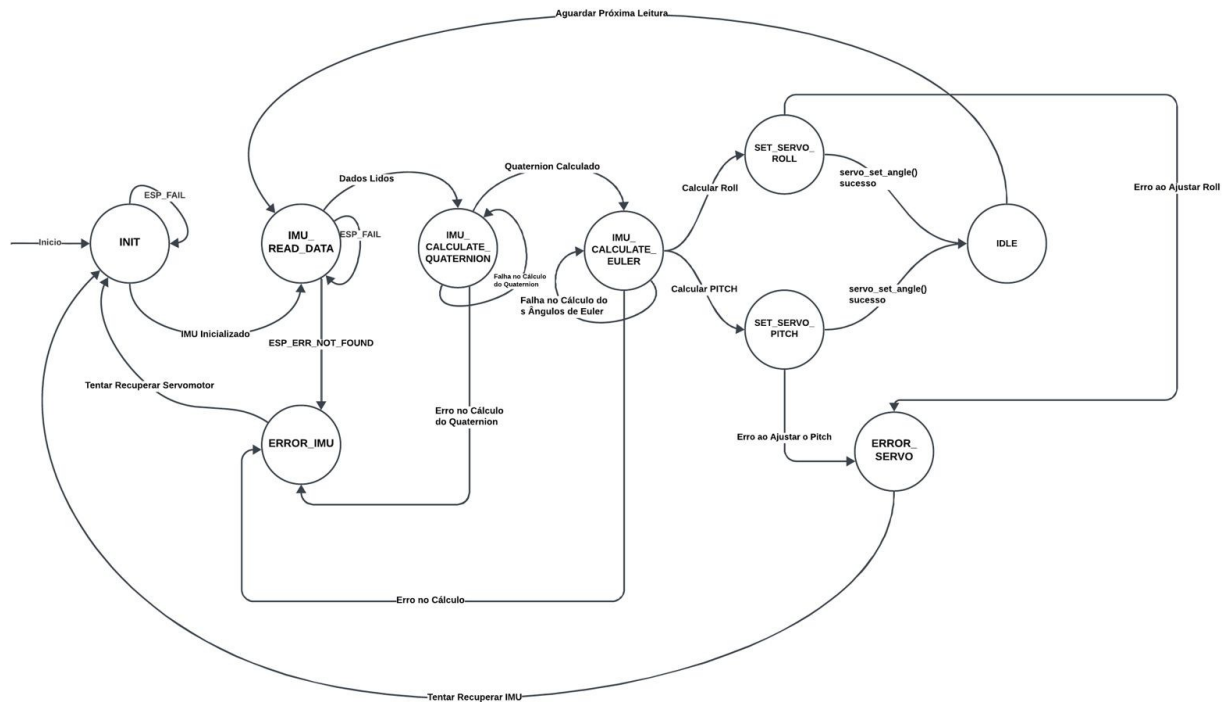
## Esquemático do hardware



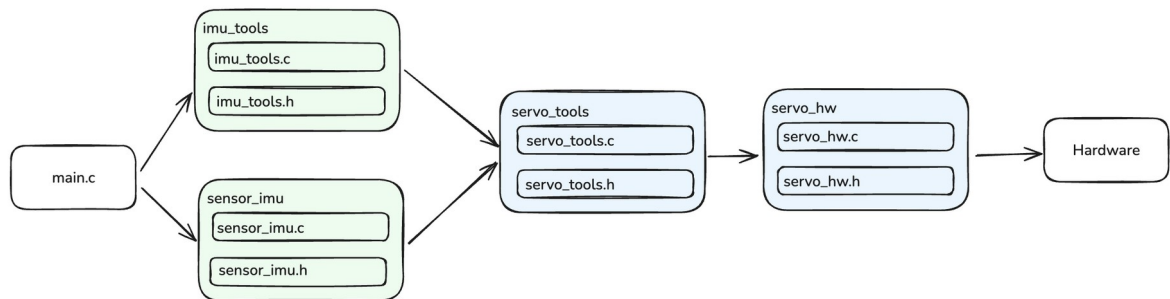
## Diagrama de Blocos



## Máquina de Estados



## Estrutura do Projeto



```

|— build/
|   # Pasta onde os arquivos compilados são armazenados
|
|— components/
|   |— servo_tools/
|   |   |— include/
|   |   |   |— servo_tools.h # Arquivo de cabeçalho para as funções
de controle do servo
|   |   |— servo_tools.c      # Arquivo fonte para as funções de
controle do servo
|   |   |— CMakeLists.txt    # Arquivo CMake específico para o

```

```

componente servo_tools
├── servo_hw/
│   ├── include/
│   │   └── servo_hw.h      # Arquivo de cabeçalho para a abstração
de hardware do servo
│   ├── servo_hw.c          # Arquivo fonte para a abstração de
hardware do servo
│   └── CMakeLists.txt      # Arquivo CMake específico para o
componente servo_hw
├── docs/
│   ├── documentation.pdf   # PDF com documentação para firmware e
bibliotecas
│   ├── state_machine.png   # Diagrama de máquina de estados para
firmware
│   └── circuit_diagram.jpg  # Diagrama de circuito para firmware
├── main/
│   ├── CMakeLists.txt      # Arquivo CMake para a aplicação
principal
│   └── main.c              # Arquivo fonte da aplicação principal
├── .gitignore              # Arquivo que especifica quais arquivos
ou diretórios devem ser ignorados pelo Git
├── CMakeLists.txt          # Arquivo CMake principal para todo o
projeto
└── sdkconfig               # Arquivo de configuração
(gerado/gerenciado pelo "make menuconfig")

```

## Executar projeto

1. Clone o repositório para sua máquina local:

```

```bash
git clone https://github.com/jmmarcoss/projeto-embarcados-3.git
```

```

1. Navegue até o diretório da etapa 1 projeto:

```

```bash
cd ./projeto-embarcados-3
```

```

1. Abra o ESP-IDF
2. Selecione: **Configure ESP-IDF Extension**
3. Selecione: **Use Existing Setup**

4. Selecione: **Advanced / Add .vscode subdirectory files**
5. Selecione: **Build**
6. Feche o ESP-IDF
7. Verifique o status do build

## Bibliotecas

### servo\_tools

#### Tipos de Dados

##### ServoConfig

A estrutura `ServoConfig` é usada para configurar os parâmetros de controle de um servo motor.

```
typedef struct
{
    uint8_t gpio_num;
    uint32_t pwm_freq;
    ServoAngle min_angle;
    ServoAngle max_angle;
    uint32_t min_pulse_width;
    uint32_t max_pulse_width;
} ServoConfig;
```

#### Atributos:

- `gpio_num`: Variável do tipo `uint8_t`, que representa o pino GPIO conectado ao servo.
- `pwm_freq`: Variável do tipo `uint32_t`, que representa a frequência PWM utilizada para controlar o servo.
- `min_angle`: Variável do tipo `ServoAngle`, que representa o ângulo mínimo permitido para o movimento do servo.
- `max_angle`: Variável do tipo `ServoAngle`, que representa o ângulo máximo permitido para o movimento do servo.
- `min_pulse_width`: Variável do tipo `uint32_t`, que representa a largura mínima do pulso em microssegundos.
- `max_pulse_width`: Variável do tipo `uint32_t`, que representa a largura máxima do pulso em microssegundos.

## ServoAngle

ServoAngle é um tipo `uint16_t` que representa o ângulo de um servo motor, geralmente em graus ou milésimos de grau.

---

```
typedef uint16_t ServoAngle;
```

## Funções

### `servo_init`:

Inicializa o servomotor com base na configuração fornecida (pino GPIO, frequência PWM, etc.).

```
esp_err_t servo_init(ServoConfig *config);
```

### Parâmetros:

- **config**: Ponteiro para uma estrutura `ServoConfig` que contém as configurações do servomotor.
  - **Tipo**: `ServoConfig *`
  - **Descrição**: Deve apontar para uma estrutura `ServoConfig` válida que especifica os parâmetros de configuração, como pino GPIO e frequência PWM.

### Valor Retornado:

- **Tipo**: `esp_err_t`
- **Valores**:
  - `ESP_OK`: Sucesso na inicialização.
  - `ESP_ERR_INVALID_ARG`: Falha na inicialização devido a argumentos inválidos.

## Exemplo de Uso

```
#include <stdio.h>
#include "esp_err.h"
#include "servo_tools.h"

int main(void) {
    ServoConfig config = {
        .gpio_pin = 18,
        .pwm_freq = 50,
        .min_pulse_width = 1000,
        .max_pulse_width = 2000
    };

    esp_err_t result_servo = servo_init(&config);

    if (result_servo == ESP_OK) {
        printf("Servomotor inicializado com sucesso.\n");
    }
}
```

```

    } else {
        printf("Falha na inicialização do servomotor. Código do erro:
%d\n", result_servo);
    }

    return 0;
}

```

### **servo\_set\_angle:**

Configura o ângulo do servomotor com base na configuração fornecida.

```
esp_err_t servo_set_angle(ServoConfig *config, ServoAngle angle);
```

### **Parâmetros:**

- **config:**
  - **Tipo:** ServoConfig \*
  - **Descrição:** Ponteiro para a estrutura ServoConfig que contém a configuração do servomotor.
- **quaternion:**
  - **Tipo:** ServoAngle \*
  - **Descrição:** Ângulo desejado para o servomotor.

### **Valor Retornado:**

- ESP\_OK: Sucesso na configuração do ângulo.
- ESP\_FAIL: Falha na configuração do ângulo.

```

#include <stdio.h>
#include "esp_err.h"
#include "servo_tools.h"

int main(void) {
    ServoConfig config = {
        .min_pulse_width = 500,
        .max_pulse_width = 2500,
        .max_angle = 180
    };

    ServoAngle angle = 90;

    esp_err_t result = servo_set_angle(&config, angle);

    if (result == ESP_OK) {

```

```

        printf("Ângulo do servomotor ajustado para %d graus.\n",
angle);
    } else {
        printf("Erro ao ajustar o ângulo do servomotor. Código do
erro: %d\n", result);
    }

    return 0;
}

```

## servo\_get\_angles:

Obtém o ângulo atual do servomotor com base na configuração fornecida.

```

esp_err_t servo_get_angle(const ServoConfig *config, ServoAngle
*angle);

```

## Parâmetros

- **config:**
  - **Tipo:** `const ServoConfig *`
  - **Descrição:** Ponteiro constante para a estrutura `ServoConfig` que contém a configuração do servomotor.
- **angle:**
  - **Tipo:** `ServoAngle *`
  - **Descrição:** Ponteiro para uma variável do tipo `ServoAngle` onde o ângulo atual do servomotor será armazenado.

## Valor Retornado

- **ESP\_OK:** Sucesso na obtenção do ângulo.
- **ESP\_FAIL:** Falha na obtenção do ângulo.

## Exemplo de Uso

```

#include <stdio.h>
#include "esp_err.h"
#include "servo_tools.h"

int main(void) {
    ServoConfig config = {
        .min_pulse_width = 500,
        .max_pulse_width = 2500,
        .max_angle = 180
    };

    ServoAngle current_angle;

    esp_err_t result = servo_get_angle(&config, &current_angle);
}

```



```

        if (result == ESP_OK) {
            printf("Ângulo atual do servomotor: %d graus.\n",
current_angle);
        } else {
            printf("Erro ao obter o ângulo do servomotor. Código do erro:
%d\n", result);
        }

        return 0;
    }
}

```

## servo\_hw

### Funções

#### hw\_servo\_init

Inicializa o servomotor e aloca os recursos necessários.

```
esp_err_t hw_servo_init(uint8_t gpio_num);
```

#### Valor Retornado:

- **ESP\_OK:** Sucesso na inicialização.
- **ESP\_FAIL:** Erro durante a inicialização.

#### Parâmetros:

- **gpio\_num:**
  - **Tipo:** `uint8_t`
  - **Descrição:** Número do GPIO onde o PWM será habilitado.

#### Exemplo de Uso

```

#include <stdio.h>
#include "esp_err.h"
#include "servo_hw.h"

int main(void) {
    uint8_t gpio_num = 18;
    esp_err_t result = hw_servo_init(gpio_num);

    if (result == ESP_OK) {
        printf("Servomotor inicializado com sucesso no GPIO %d.\n",
gpio_num);
    } else {
        printf("Erro ao inicializar o servomotor no GPIO %d. Código do
erro: %d\n", gpio_num, result);
    }
}

```

```
    return 0;
}
```

### hw\_servo\_set\_pulse\_width:

Define a largura de pulso para o controle do ângulo do servo.

```
esp_err_t hw_servo_set_pulse_width(uint8_t gpio_num, uint32_t
pulse_width_us);
```

#### Parâmetros:

- **gpio\_num:**
  - **Tipo:** `uint8_t`
  - **Descrição:** Número do GPIO onde o PWM está habilitado.
- **pulse\_width\_us:**
  - **Tipo:** `uint32_t`
  - **Descrição:** Largura do pulso em microssegundos (us) a ser definida para o controle do ângulo do servo.

#### Valor Retornado:

- **ESP\_OK:** Sucesso ao definir a largura de pulso.
- **ESP\_FAIL:** Erro ao definir a largura de pulso.

#### Exemplo de Uso

```
#include <stdio.h>
#include "esp_err.h"
#include "servo_hw.h"

int main(void) {
    uint8_t gpio_num = 18;
    uint32_t pulse_width_us = 1500;

    esp_err_t init_result = hw_servo_init(gpio_num);
    if (init_result != ESP_OK) {
        printf("Erro ao inicializar o servomotor no GPIO %d. Código do
erro: %d\n", gpio_num, init_result);
        return init_result;
    }

    esp_err_t set_pulse_result = hw_servo_set_pulse_width(gpio_num,
pulse_width_us);
    if (set_pulse_result == ESP_OK) {
        printf("Largura de pulso definida para %d microssegundos no
GPIO %d.\n", pulse_width_us, gpio_num);
    } else {
```

```

        printf("Erro ao definir a largura de pulso no GPIO %d. Código
do erro: %d\n", gpio_num, set_pulse_result);
    }

    hw_servo_deinit(gpio_num);

    return 0;
}

```

#### hw\_servo\_deinit:

Desinicializa o servomotor e libera os recursos alocados.

```
esp_err_t hw_servo_deinit(uint8_t gpio_num);
```

#### Parâmetros:

- **gpio\_num:**
  - **Tipo:** uint8\_t
  - **Descrição:** Número do GPIO onde o PWM será desabilitado.

#### Valor Retornado:

- **ESP\_OK:** Sucesso na desinicialização.
- **ESP\_FAIL:** Erro durante a desinicialização.

#### Exemplo de Uso

```

#include <stdio.h>
#include "esp_err.h"
#include "servo_tools.h"

int main(void) {
    ServoConfig config = {
        .min_pulse_width = 500,
        .max_pulse_width = 2500,
        .max_angle = 180
    };

    esp_err_t result = hw_servo_deinit(&config);

    if (result == ESP_OK) {
        printf("Hardware do servomotor desinicializado com sucesso.\n");
    } else {
        printf("Erro ao desinicializar o hardware do servomotor.
Código do erro: %d\n", result);
    }

    return 0;
}

```

```
}
```

## sensor\_imu

### Tipos de Dados

#### AccelerationData

A estrutura `AccelerationData` é usada para armazenar os dados de aceleração obtidos a partir do sensor IMU MPU6050.

```
typedef struct {  
    float accel_X;  
    float accel_Y;  
    float accel_Z;  
} AccelerationData;
```

#### Atributos:

- `accel_X`: Valor de aceleração x (g)
- `accel_Y`: Valor de aceleração y (g)
- `accel_Z`: Valor de aceleração z (g)

#### Observação:

Todos os valores de aceleração (x/y/z) usam unidades de força g, onde  $1g = 9,80665 \text{ m/s}^2$ .

#### GyroscopeData

A estrutura `GyroscopeData` é usada para armazenar os dados de rotação obtidos a partir do sensor IMU MPU6050.

```
typedef struct {  
    int16_t rotation_X;  
    int16_t rotation_Y;  
    int16_t rotation_Z;  
} GyroscopeData;
```

#### Atributos:

- `rotation_X`: Valor de rotação x (deg/sec)
- `rotation_Y`: Valor de rotação y (deg/sec)
- `rotation_Z`: Valor de rotação z (deg/sec)

#### Observação:

O giroscópio mede a rotação angular, fornecendo a taxas de rotação (x/y/z) em graus por segundo.

## Funções

### `imu_init`

Esta função é responsável por inicializar o sensor inercial e prepará-lo para a obtenção dos dados.

```
esp_err_t imu_init(uint8_t devAddr, gpio_num_t sda_pin, gpio_num_t scl_pin);
```

#### Valor Retornado:

- **ESP\_OK:** Sucess
- **ESP\_ERR\_NOT\_FOUND:** Initialization failure

#### Parâmetros:

- **devAddr:**
  - **Tipo:** uint8\_t
  - **Descrição:** Endereço do dispositivo do sensor IMU.
- **sda\_pin:**
  - **Tipo:** gpio\_num\_t
  - **Descrição:** Número do pino GPIO usado para comunicação SDA (data).
- **scl\_pin:**
  - **Tipo:** gpio\_num\_t
  - **Descrição:** Número do pino GPIO usado para comunicação SCL (clock).

#### Exemplo de Uso

```
#include <stdio.h>
#include "esp_err.h"
#include "sensor_imu.h"

#define MPU6050_ADDR 0x68
#define SDA_PIN GPIO_NUM_21
#define SCL_PIN GPIO_NUM_22

int main(void) {
    esp_err_t result_imu = imu_init(MPU6050_ADDR, SDA_PIN, SCL_PIN);
    if (result_imu != ESP_OK)
    {
        printf("Failed to init IMU Sensor: %s\n",
        esp_err_to_name(result_imu));
        return;
    }
}
```

```
    return 0;
}
```

Função `imu_get_acceleration_data`:

Esta função permite obter os dados de aceleração do sensor inercial e armazená-los na estrutura `AccelerationData` fornecida

```
esp_err_t imu_get_acceleration_data(AccelerationData *data):
```

**Parâmetros:**

- **data:**
  - **Tipo:** `AccelerationData *`
  - **Descrição:** Ponteiro para a estrutura `AccelerationData` onde os dados de aceleração serão armazenados.

**Valor Retornado:**

- **ESP\_OK:** Sucess
- **ESP\_FAIL:** Initialization failure

**Exemplo de Uso**

```
#include <stdio.h>
#include "esp_err.h"
#include "sensor_imu.h"

int main() {
    AccelerationData data;

    esp_err_t result_accel = imu_get_acceleration_data(&data);

    if (result_accel == ESP_OK) {
        printf("Aceleração: X = %fg, Y = %fg, Z = %fg\n", data.accel_X,
            data.accel_Y, data.accel_Z);
    } else {
        printf("Falha ao ler dados de aceleração. Código de erro: %d\n",
            result_accel);
    }

    return 0;
}
```

Função `imu_get_gyroscope_data`:

Esta função permite obter os dados do giroscópio do sensor inercial e armazená-los na estrutura `GyroscopeData` fornecida como parâmetro.

```
esp_err_t imu_get_gyroscope_data(GyroscopeData *data):
```

#### Parâmetros:

- **data:**
  - **Tipo:** GyroscopeData \*
  - **Descrição:** Ponteiro para a estrutura GyroscopeData, onde os dados do giroscópio serão armazenados.

#### Valor Retornado:

- **ESP\_OK:** Sucess
- **ESP\_FAIL:** Initialization failure

#### Exemplo de Uso

```
#include <stdio.h>
#include "esp_err.h"
#include "sensor_imu.h"

int main(void) {
    GyroscopeData data;

    esp_err_t result_gyro = imu_get_gyroscope_data(&data);

    if (result_gyro == ESP_OK) {
        printf("Giroscópio: (X: %d°/sec, Y: %d°/sec, Z: %d°/sec)\n",
data.rotation_X, data.rotation_Y, data.rotation_Z);
    } else {
        printf("Falha ao ler dados do giroscópio. Código de erro: %d\n",
result_gyro);
    }

    return 0;
}
```

#### Função imu\_deinit:

Esta função desabilita a comunicação I2C e libera os recursos associados à comunicação com o sensor IMU.

```
esp_err_t imu_deinit():
```

#### Parâmetros:

- Esta função não recebe parâmetros. Ela é declarada com **void**, indicando que não são necessários argumentos para sua execução.

#### Valor Retornado:

- ESP\_OK: Sucess
- ESP\_FAIL: Initialization failure

### Exemplo de Uso

```
#include <stdio.h>
#include "esp_err.h"
#include "sensor_imu.h"

int main(void) {
    imu_deinit()

    return 0;
}
```

## imu\_tools

### Tipos de Dados

#### IMUData

A estrutura `IMUData` é usada para armazenar os dados de aceleração e giroscópio obtidos a partir do sensor IMU MPU6050.

```
typedef struct
{
    AccelerationData accelData;
    GyroscopeData gyroData;
} IMUData;
```

#### Atributos:

- `accelData`: Variável do tipo `AccelerationData`, que representa os dados de aceleração.
- `gyroData`: Variável do tipo `GyroscopeData`, que representa os dados de rotação do giroscópio.

#### Quaternion

A estrutura `Quaternion` é utilizada para representar orientações em três dimensões.

---

```
typedef struct
{
    float w;
    float x;
    float y;
```



```
float z;  
} Quaternion;
```

- **w**: Componente escalar do quaternion.
  - Descrição: Representa o componente real de um quaternion.
- **x**: Componente do eixo x do quaternion.
  - Descrição: Representa a primeira componente imaginária do quaternion.
- **y**: Componente do eixo y do quaternion.
  - Descrição: Representa a segunda componente imaginária do quaternion.
- **z**: Componente do eixo z do quaternion.
  - Descrição: Representa a terceira componente imaginária do quaternion.

## EulerAngles

A estrutura **EulerAngles** é utilizada para armazenar os ângulos de Euler, que descrevem a orientação de um objeto no espaço tridimensional. Os ângulos de Euler são uma forma conveniente de expressar rotações sequenciais em torno dos eixos principais (Z, Y, X), conhecidos como yaw, pitch e roll, respectivamente.

---

```
typedef struct  
{  
    float roll;  
    float pitch;  
    float yaw;  
} EulerAngle;
```

- **roll**:
  - **Descrição**: Ângulo de rotação em torno do eixo X do sistema de coordenadas. Utilizado principalmente para descrever a inclinação lateral de um objeto, como a inclinação das asas de uma aeronave.
- **pitch**:
  - **Descrição**: Ângulo de rotação em torno do eixo Y do sistema de coordenadas. Usado para descrever o ângulo de inclinação frontal de um objeto, como o nariz de uma aeronave apontando para cima ou para baixo.
- **yaw**:
  - **Descrição**: Ângulo de rotação em torno do eixo Z do sistema de coordenadas. Utilizado para descrever a orientação horizontal de um objeto, como a direção para a qual a frente de um carro ou aeronave está apontando.

## Funções

### **imu\_read\_data**:

Esta função obtém os dados do sensor IMU e os armazena na estrutura **IMUData** fornecida como parâmetro

```
esp_err_t imu_read_data(IMUData *data);
```

#### Parâmetros:

- **data:** Ponteiro para a estrutura IMUData onde os dados do IMU serão armazenados.
  - **Tipo:** IMUData \*
  - **Descrição:** Deve apontar para uma estrutura IMUData válida que será preenchida com os valores de aceleração e giroscópio.

#### Valor Retornado:

- **Tipo:** esp\_err\_t
- **Valor:**
  - ESP\_OK: Success
  - ESP\_FAIL: Initialization failure.

#### Exemplo de Uso

```
#include <stdio.h>
#include "esp_err.h"
#include "imu_tools.h"

int main(void) {

    IMUData data = {
        .accel_X = 0.0, .accel_Y = 0.0, .accel_Z = 1.0,
        .rotation_X = 0, .rotation_Y = 0, .rotation_Z = 0,
    };

    esp_err_t result_IMU = imu_read_data(IMUData &data)

    if (result_IMU == ESP_OK) {
        printf("IMU Data: \n Aceleração: X = %fg, Y = %fg, Z = %fg \n
Giroscópio: (X: %d°/sec, Y: %d°/sec, Z: %d°/sec)\n",
        data.accel_X, data.accel_Y, data.accel_Z,
        data.rotation_X, data.rotation_Y, data.rotation_Z);
    } else {
        printf("Falha ao ler os dados do sensor. Código do erro: %d\
n", result_IMU);
    }

    return 0;
}
```

#### imu\_calculate\_quaternion:

Esta função calcula o quaternion com base nos dados do sensor IMU fornecidos e armazena o resultado na estrutura Quaternion.

```
esp_err_t imu_calculate_quaternion(const IMUData *data, Quaternion *quaternion);
```

#### Parâmetros:

- **data:**
  - **Tipo:** IMUData \*
  - **Descrição:** Ponteiro para a estrutura IMUData, onde os dados do IMU serão armazenados.
- **quaternion:**
  - **Tipo:** Quaternion \*
  - **Descrição:** Ponteiro para a estrutura Quaternion, onde os dados do Quaternion serão armazenados.

#### Valor Retornado:

- ESP\_OK: Success
- ESP\_FAIL: Initialization failure.

#### Exemplo de Uso

```
#include <stdio.h>
#include "esp_err.h"
#include "imu_tools.h"

int main(void) {

    IMUData data = {
        .accel_X = 0.0, .accel_Y = 0.0, .accel_Z = 1.0,
        .gyro_x = 0, .gyro_y = 0, .gyro_z = 0,
    };

    Quaternion quaternion;

    esp_err_t result_quaternion = imu_calculate_quaternion(&data,
&quaternion);

    if (result_quaternion == ESP_OK) {
        printf("Quaternion: (w: %.2f, x: %.2f, y: %.2f, z: %.2f)\n",
            quaternion.w, quaternion.x, quaternion.y,
quaternion.z);
    } else {
        printf("Erro ao calcular quaternion. Código do erro: %d\n",
result_quaternion);
    }
}
```

```
    return 0;
}
```

### imu\_calculate\_euler\_angles:

Esta função converte o quaternion em ângulos de Euler com base nos dados do sensor IMU fornecidos e armazena o resultado na estrutura EulerAngle.

```
esp_err_t imu_calculate_euler_angles(const Quaternion *quaternion,
EulerAngle *euler);
```

#### Parâmetros:

- **quaternion:**
  - **Tipo:** Quaternion \*
  - **Descrição:** Ponteiro para a estrutura Quaternion, onde os dados do Quaternion serão armazenados.
- **euler:**
  - **Tipo:** EulerAngle \*
  - **Descrição:** Ponteiro para a estrutura EulerAngle, onde os dados do EulerAngle serão armazenados.

#### Valor Retornado:

- ESP\_OK: Success
- ESP\_FAIL: Initialization failure.

#### Exemplo de Uso

```
#include <stdio.h>
#include "esp_err.h"
#include "imu_tools.h"

int main(void) {
    Quaternion quaternion = {
        .w = 1.0, .x = 0.0, .y = 0.0, .z = 0.0
    };

    EulerAngle euler;

    esp_err_t result_euler = imu_calculate_euler_angles(const
Quaternion *quaternion, EulerAngle *euler);

    if (result_euler == ESP_OK) {
```

```
        printf("Euler Angles: (roll: %.2f, pitch: %.2f, yaw: %.2f)\n",
               euler.roll, euler.pitch, euler.yaw);
    } else {
        printf("Erro ao calcular ângulos de Euler. Código do erro: %d\n", result_euler);
    }

    return 0;
}
```

## Considerações finais sobre as libs dos grupos

- As libs a principio nao estavam seguindo o padrao ESP-IDF, entramos em contato com os grupos e fizeram as devidas correções.
- O arquivo Cmake não estavam buildando e então tivemos que ajustar para indexar todos os arquivos.
- Os 2 grupos não seguiram os padrões da máquina de estado.