Projeto Sistemas Embarcados

Projeto Sistemas Embarcados

Etapa 1 - Biblioteca para utilizar os dados do acelerômetro e do giroscópio do sensor inercial MPU6050.

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**

Firmware

Esquemático do hardware

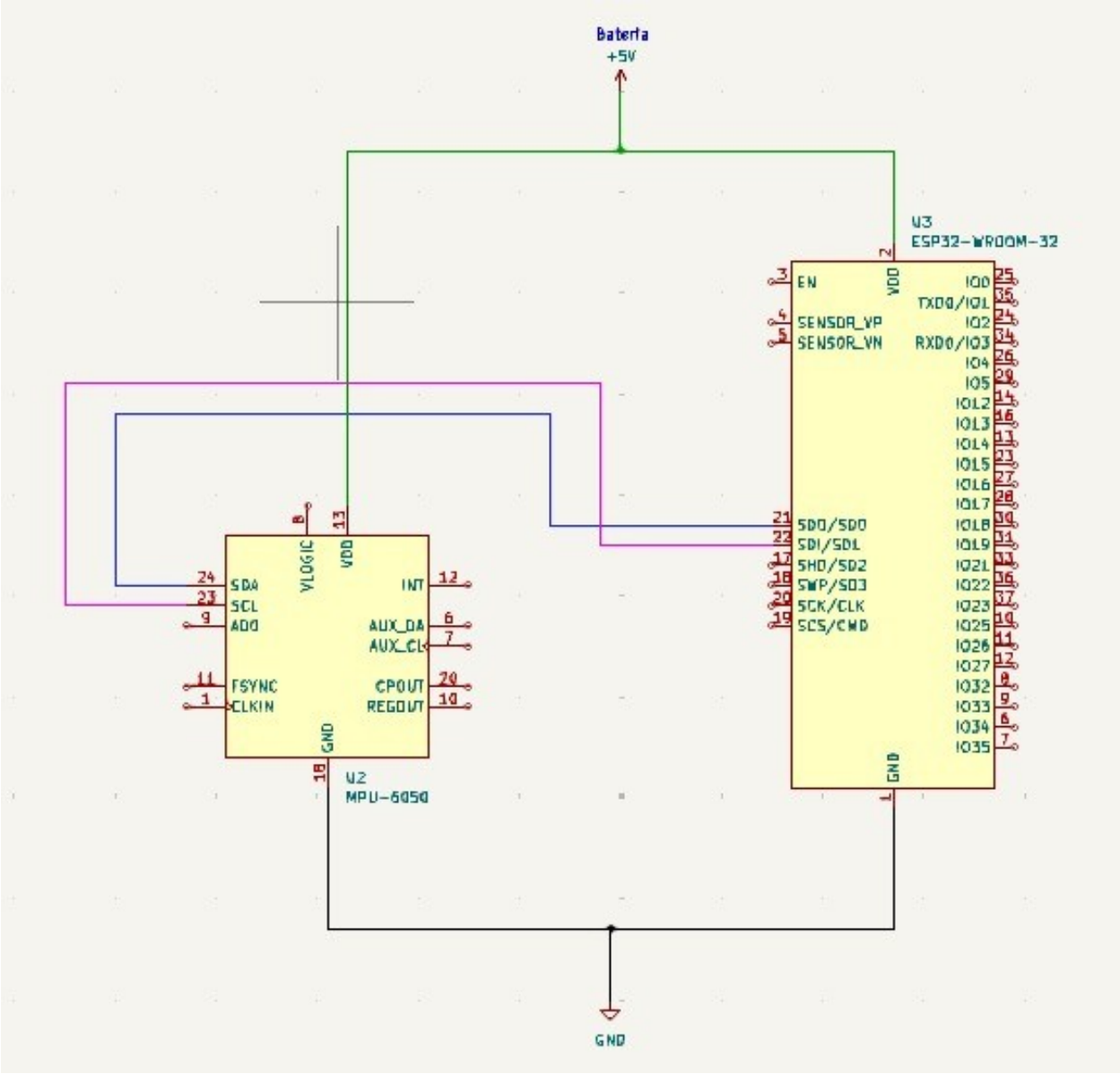
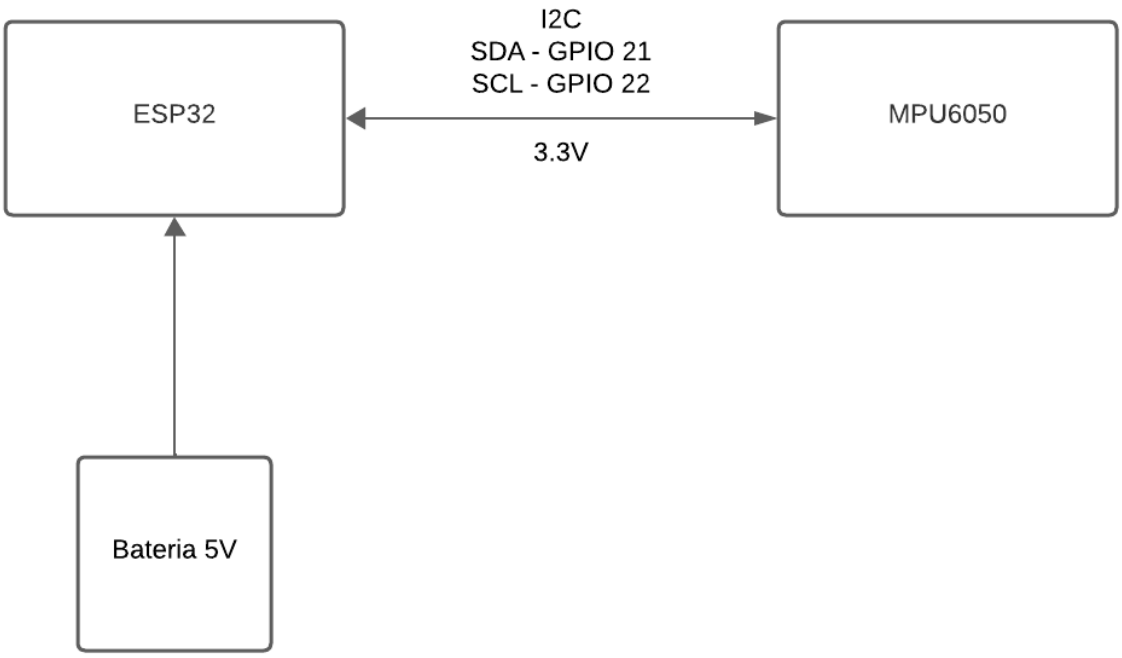
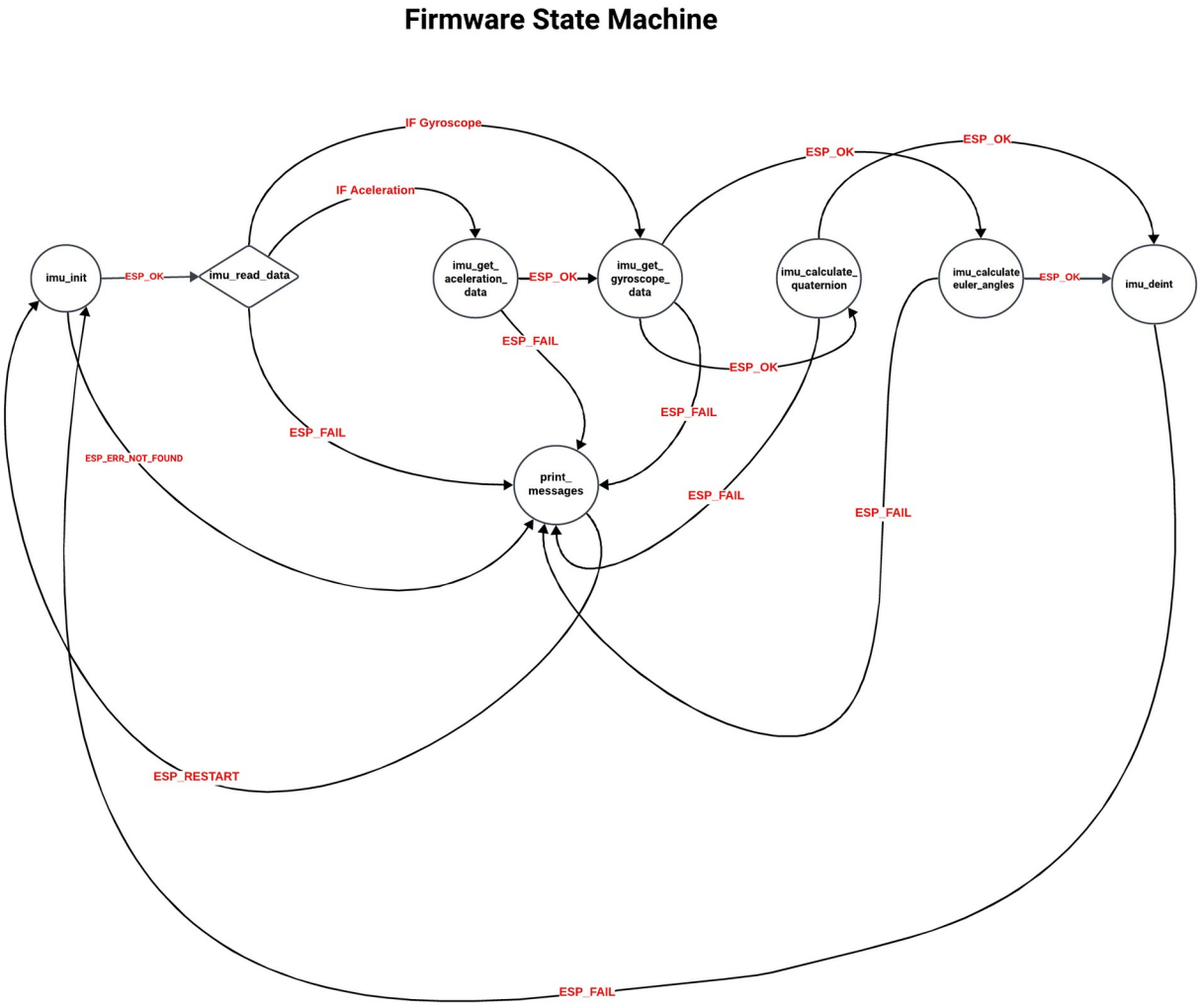


Diagrama de Blocos



Máquina de Estados



Estrutura do Projeto

```
etapa-1-servo/
│
├── build/                          # Pasta onde os arquivos compilados
são armazenados
│
├── components/                     # Diretório para componentes
personalizados ou de terceiros
│   ├── imu_tools/                  # Componente para ferramentas IMU
│   │   ├── include/               # Arquivos de cabeçalho para o
componente IMU tools
│   │   │   └── imu_tools.h        # Arquivo de cabeçalho para as funções
IMU tools
│   │   ├── imu_tools.c            # Arquivo fonte para as funções IMU
tools
```

```
│   │   └── CMakeLists.txt      # Arquivo CMake específico para o
componente IMU tools
│   │
│   └── sensor_imu/             # Componente para sensor IMU
│       ├── include/           # Arquivos de cabeçalho para o
componente sensor IMU
│       │   └── sensor_imu.h    # Arquivo de cabeçalho para as funções
sensor IMU
│       ├── sensor_imu.c        # Arquivo fonte para as funções sensor
IMU
│       └── CMakeLists.txt      # Arquivo CMake específico para o
componente sensor IMU
│
├── docs/                       # Diretório contendo documentação
relacionada ao projeto
│   ├── documentation.pdf       # Documentação para firmware e
bibliotecas
│   ├── state_machine.png       # Diagrama de máquina de estados para
firmware
│   └── circuit_diagram.jpg     # Diagrama de circuito para firmware
│
├── main/                       # Código fonte principal do seu
projeto
│   ├── 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:

```
git clone https://github.com/pedromacedol/projetos-sistemas-
embarcados.git
```

2. Navegue até o diretório da etapa 1 projeto:

```
cd projetos-sistemas-embarcados/etapa-1-imu
```

3. Abra o ESP-IDF

4. Selecione: **Configure ESP-IDF Extension**
5. Selecione: **Use Existing Setup**
6. Selecione: **Advanced / Add .vscode subdirectory files**
7. Selecione: **Build**
8. Feche o ESP-IDF
9. Verifique o status do build

Bibliotecas

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.

EulerAngle

A estrutura `EulerAngle` 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;
}
```

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 m/s².

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;
}
```