

ORBISAT

Planejamento e Desenvolvimento de um cubesat voltado para 2º Etapa do [EVENTO QUE ESQUICI O NOME KKKKK]

Sumário

Introdução	4
Metodologia.....	4
Arduino IDE	4
Wokwi	4
Resultados.....	4
Eletrônica	5
Acelerômetro	5
Barômetro.....	5
Sensor de Corrente	5
Cartão Micro SD	5
Elétrica	6
Sensor MPU6050 - Acelerômetro com Giroscópio.....	6
Sensor BMP280 - Barômetro	7
Sensor ACS712 – Sensor de Corrente	7
Módulo de Cartão Micro SD	8
ESP32	8
Pacotes de Dados – Placas do Modelo ESP32	9
Integração da Arduino IDE com ESP32	10
Bibliotecas dos Sensores.....	11
Programação.....	12
Programação do Sensor MPU6050.....	12
Programação do Sensor BMP280	17
Programação do Sensor ACS712	19
Programação do Módulo de Cartão Micro SD	19
Discussão	21
Problemas com ESP32	21
Problemas com Porta USB	21
Problemas com MPU6020	22
Problemas com BMP280.....	23
Problemas com ACS712	24
Problemas com Módulo de Cartão Micro SD	24
Conclusão.....	24
Recomendações.....	25

Referências	25
-------------------	----

Introdução

Mediante aos avanços na etapa anterior, iniciamos o desenvolvimento com afinco e dedicação em prosseguirmos e obtermos excelentes resultados na 2ª etapa. Demos então continuidade assim ao projeto desenvolvendo sua parte física, tendo nesta parte do relatório informar como foi feita a parte eletrônica do projeto.

Com recursos computacionais em nível de hardware e software, desenvolvemos tanto a parte de molduras à eletrônica e programação, utilizando ao máximo os recursos que nos foram ofertados e encontrados em nossas pesquisas.

Metodologia

A utilização de softwares voltados para o **desenvolvimento e programação de microcontroladores**, instalados em um sistema operacional **Ubuntu (Linux)** e alguns utilizados através de um **navegador web**, permitiram realizar testes de programação e configurações do microcontrolador ESP32 e suas **bibliotecas** e dos **sensores e módulos** utilizados no projeto do cubesat. Para testes de hardware físicos, foi utilizado também uma **protoboard** e **jumpers** para interligar os componentes e o ESP32.

Algumas referências de pesquisa foram necessárias para entender o funcionamento de certos sensores e como os mesmos seriam conectados e energizados ao ESP32, sendo assim, as referências estarão ao final do relatório para uso diverso e pesquisa complementar, caso necessário.

Salientamos que algumas referências contêm além do escopo elétrico, alguns códigos automáticos de testes que as bibliotecas utilizam e podem/são úteis para o entendimento e modificações se necessário, a maioria dos testes usados para apresentação deste relatório serão baseados nestes códigos prontos.

Arduino IDE

É o mais popular e/ou o mais recomendado para se programar microcontroladores. Muitas vezes associada à programação do microcontrolador **Arduino**, a Arduino IDE também permite realizar a programação de microcontroladores como o **ESP32** e permite ainda por cima exportar suas bibliotecas para serem implementadas aos projetos. Para download e informações, acesse o [site oficial](#) e procure na guia de **software** para realizar download da IDE.

Wokwi

É uma **plataforma online** que permite realizar simulações de ESP32, permitindo assim desenvolver códigos e interligar componentes de maneira virtual, sem ficar preso a necessidade de ter o ESP32 e/ou seus sensores fisicamente, o que não exclui a importância de testes com o hardware físico. Para utilizar a plataforma, acesse o [site oficial](#) e caso deseje, cadastre-se para salvar seus projetos.

Resultados

Após diversos testes realizados, análises e pesquisas, consultas a pessoas com mais experiências, obtivemos êxito em grande parte do nosso projeto e falhas/pendências em outras partes. Tendo em

vista que o **ESP32** é um microcontrolador diferente do **Arduino**. Diversas incompatibilidades podem surgir ao adaptar um projeto com recursos do Arduino para o ESP 32, porém neste capítulo, abordaremos os resultados positivos, trazendo para o capítulo de **Discussão** alguns dos problemas encontrados.

Eletrônica

Em níveis de desenvolvimento de eletrônica, para realizar o monitoramento e obtenção de dados (mais precisos possível), uma solução prática, rápida, com bom custo-benefício e fácil de ser integrada é com a utilização de **sensores**.

O uso e conhecimento de sensores é algo que deve ser tratado minuciosamente, pois apesar de sua praticidade, de certo modo, há também diversos detalhes e cuidados que devem ser tomados. Felizmente após diversas pesquisas, conseguimos boas referências para lidarmos com cada sensor e cada módulo utilizado, detalhados nas referências bibliográficas, que nos permitiram ter noção de como funciona cada sensor com mais detalhes e como aplicá-los em nosso cubesat.

Para atender os requisitos do cubesat, foi realizado uma pesquisa a respeito de sensores que possuem funções necessárias para monitorar e capturar as informações que necessitamos, detalhadas mais especificamente abaixo.

Acelerômetro

É um dispositivo capaz de medir a aceleração de um objeto, permitindo saber qual a direção e sentido da aceleração do próprio objeto. Existem diversos tipos de acelerômetros, porém para o projeto foi selecionado e utilizado o **sensor MPU6050**, que além de **acelerômetro** é também um **giroscópio**, sendo assim, capaz de medir a aceleração e rotação de **3 eixos** (Eixo X, Eixo Y e Eixo Z) e **temperatura**.

Barômetro

É um instrumento utilizado mais na área de meteorologia que tem objetivos de estudo, cálculo e análises de medição de pressão atmosférica. Em níveis mais altos e específicos, existem e são utilizados os barômetros de **mercúrio** e **aneroides(metálicos)**. Porém, para se utilizar um barômetro no cubesat, foi utilizado o **sensor BMP280**, que além de ter um consumo de energia menor que outro modelos, consegue medir **pressão, altitude e temperatura**.

Sensor de Corrente

Para melhor uso da fonte de energia que alimentará o ESP32, foi necessário utilizar mecanismos de controle para que a energia não venha a prejudicar o funcionamento do ESP32 – queimando ou não fornecendo energia suficiente para seu funcionamento – assim, chegamos a conclusão de que seria necessário um controlador de corrente elétrica, para isto foi escolhido o **sensor ACS712**.

Cartão Micro SD

Com objetivo de manter a salvo todas as informações foi necessário estudar qual mecanismo e/ou melhor maneira de se armazenar todos os dados obtidos pelos sensores, tendo em vista que o

ESP32 não tem, ou melhor, não é aconselhável para uso de armazenamento de informação em **memória**.

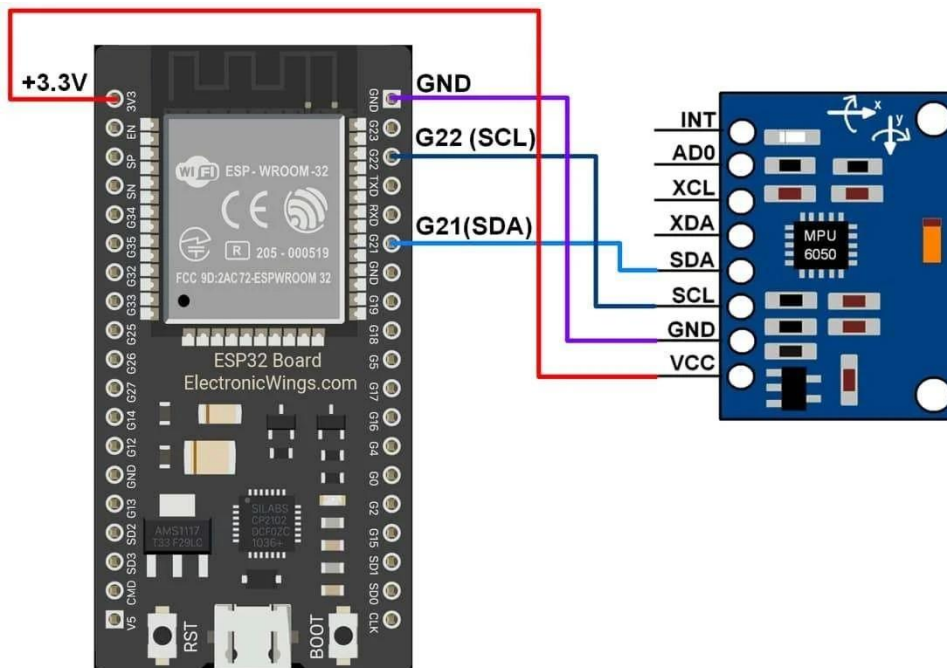
Uma solução para esta questão foi o uso de um **módulo de Cartão Micro SD**, que utilizará um cartão Micro SD para armazenar os dados e mantê-los como uma espécie de **backup** e permitir assim a **integridade e disponibilidade**, mantendo pilares importantíssimos da área de **segurança da informação**.

Elétrica

A parte elétrica é uma que temos que ter bastante cuidado pois qualquer ligação feita incorretamente, poderá ocasionar a perda do ESP32, do sensor ou ambos. Felizmente através de estudos acadêmicos isto não foi um problema por completo, houve sim dificuldades, porém foram superadas.

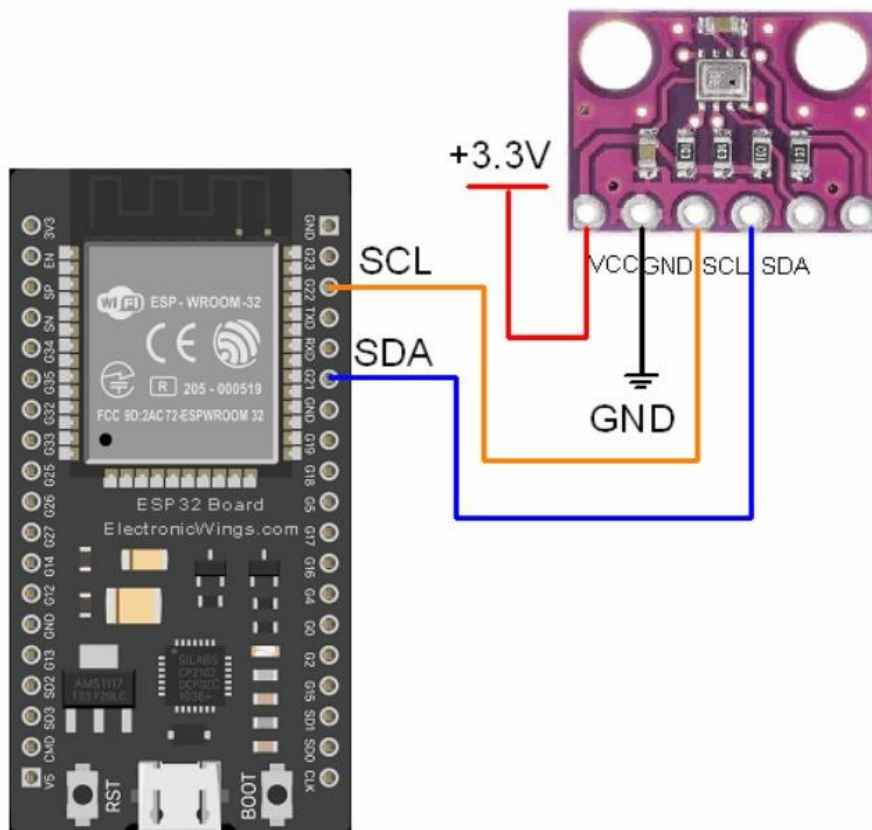
Para cada sensor, podemos apresentar seu esquema elétrico conforme encontramos em nossas pesquisas, citadas anteriormente. Estes esquemas são referentes à sensores e módulos apresentados abaixo:

Sensor MPU6050 - Acelerômetro com Giroscópio



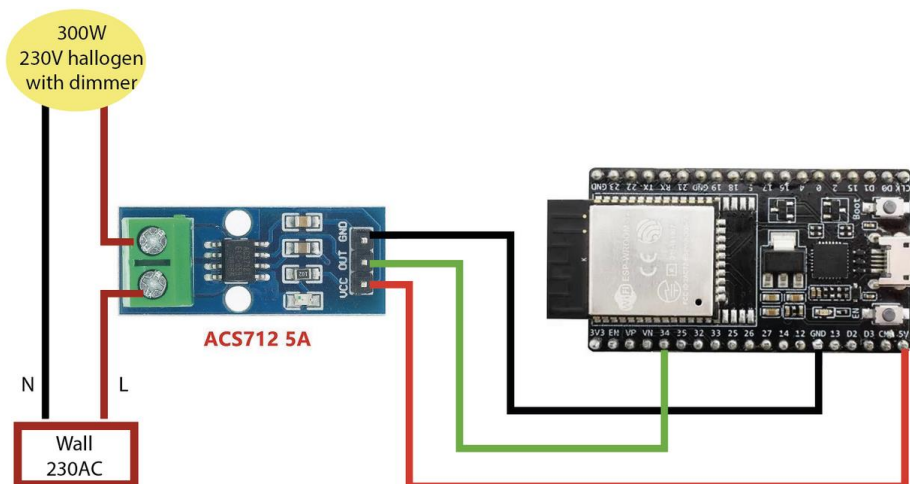
Fonte: <https://www.electronicwings.com/esp32/mpu6050-gyroscope-interfacing-with-esp32>

Sensor BMP280 - Barômetro



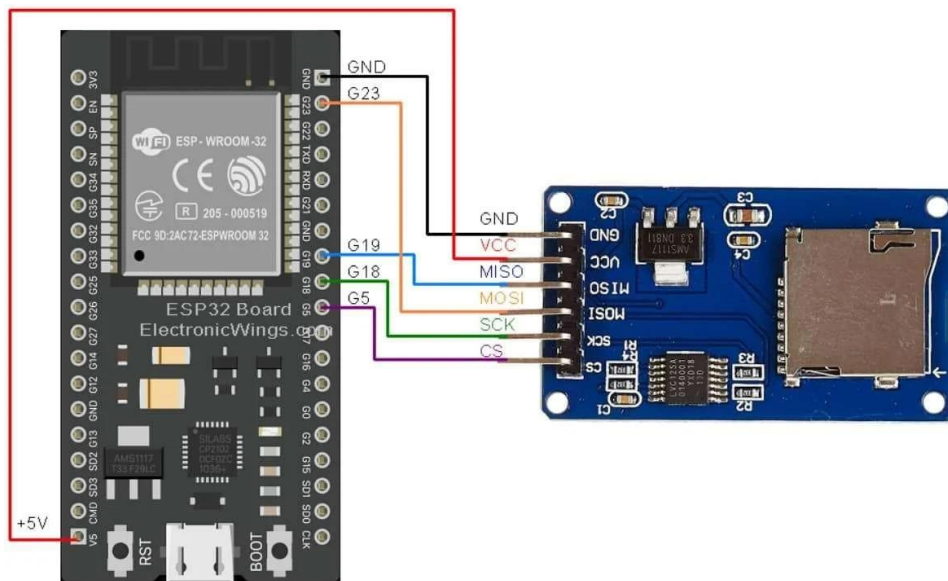
Fonte: <https://www.electronicwings.com/esp32/bmp280-barometer-sensor-interfacing-with-esp32>

Sensor ACS712 – Sensor de Corrente



Fonte: <https://postimg.cc/pp70DKSb>

Módulo de Cartão Micro SD

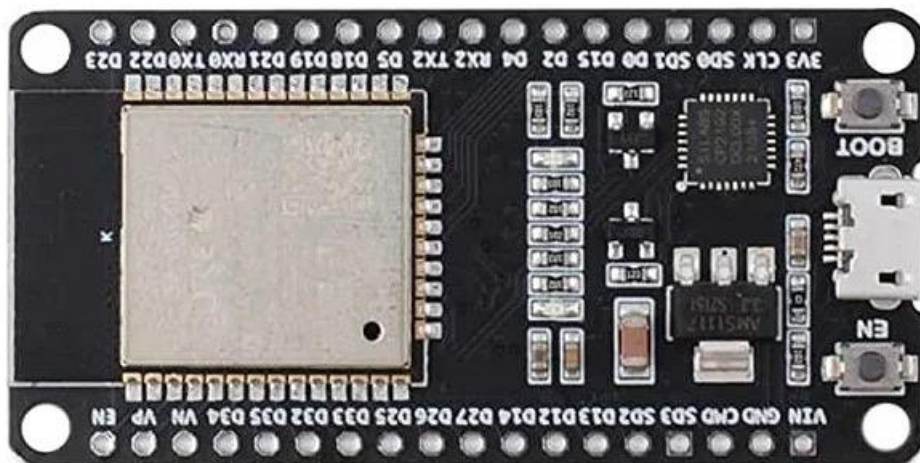


Fonte: <https://www.electronicwings.com/esp32/microsd-card-interfacing-with-esp32>

ESP32

O ESP 32 utilizado foi o ESP32 NodeMCU - DEVKIT V1, sua versatilidade e custo-benefício foram ideias para sua aquisição. Sempre recomendamos a busca e utilização pelo [site da própria fabricante](#) para informações mais precisas.

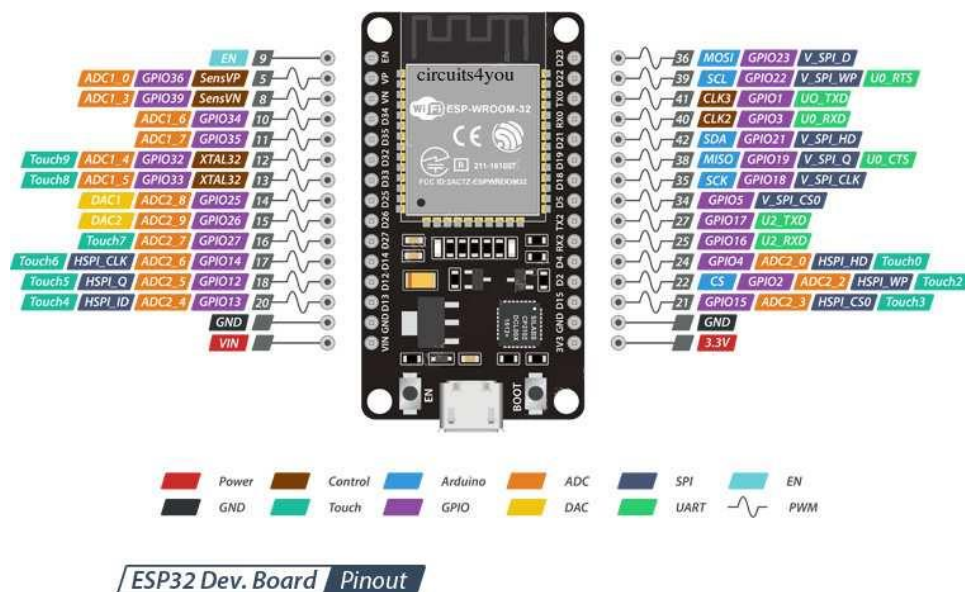
O datasheet do ESP32 utilizado pode ser localizado [aqui](#). Abaixo a imagem do ESP32 utilizado.



Fonte: <https://www.iot-robotica.com.br/produto/esp32-dev-kit.html>

Para se utilizar um ESP32 é necessário conhecer sua arquitetura e suas funcionalidades, por exemplo usando o datasheet do próprio produto, fornecido pela fabricante (citada acima), para conhecer e

saber onde e qual sensor utilizar junto do ESP32. Como referência, utilizamos o datasheet fornecido pela fabricante **Espressif Systems®** e para uma melhor visualização de funções a imagem abaixo, para conhecimento visual das funções de cada pino.



Fonte: <https://circuits4you.com/2018/12/31/esp32-devkit-esp32-wroom-gpio-pinout/>

Pacotes de Dados – Placas do Modelo ESP32

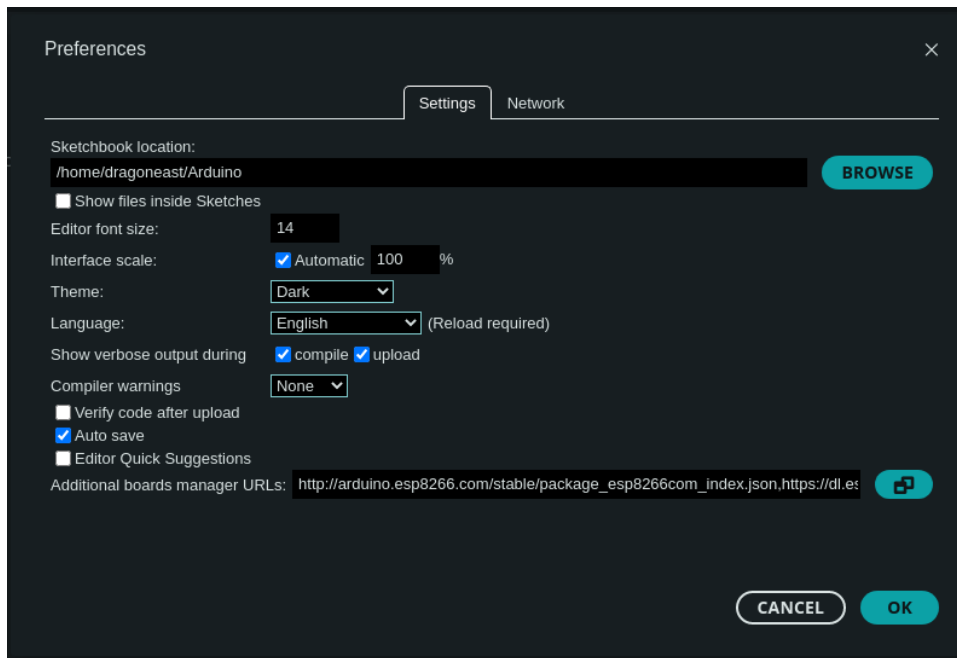
Como foi utilizada a **Arduino IDE** para a programação do ESP32, e a Arduino IDE por padrão vem apenas com configurações voltadas para programação de **Arduino**, foi necessário importar **pacotes de dados** através de URLs externas, que estão apresentadas abaixo:

http://arduino.esp8266.com/stable/package_esp8266com_index.json

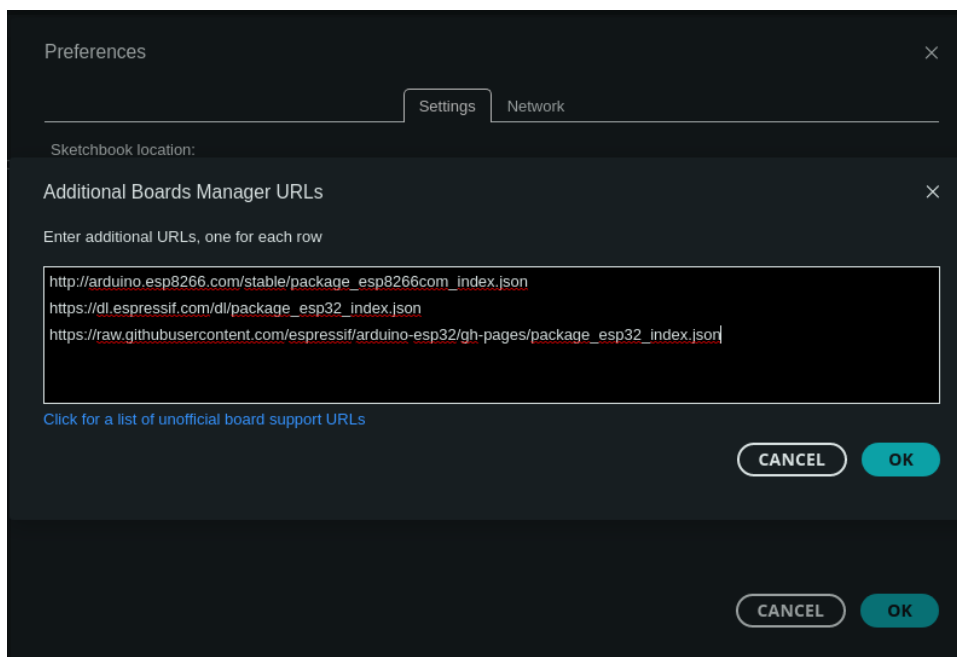
https://dl.espressif.com/dl/package_esp32_index.json

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

Estas URLs contêm pacotes de dados, ou seja, um conjunto de informações reunidas em um só lugar que realizam funções específicas ou armazenam dados específicos, que no caso das URLs acima contêm os dados e informações a respeito dos dados necessários para se programar, testar e exportar código em uma arquitetura do microcontrolador do ESP32. Para que elas sejam usadas corretamente, é necessário importar as URLs acima na guia FILE > PREFERENCES ou utilizando os comandos de atalho **Ctrl + Vírgula (,)**. Ambas as maneiras irão te direcionar para a tela apresentada abaixo.



Após chegar nesta tela, importe as URL clicando no ícone verde azulado acima do botão “OK”.

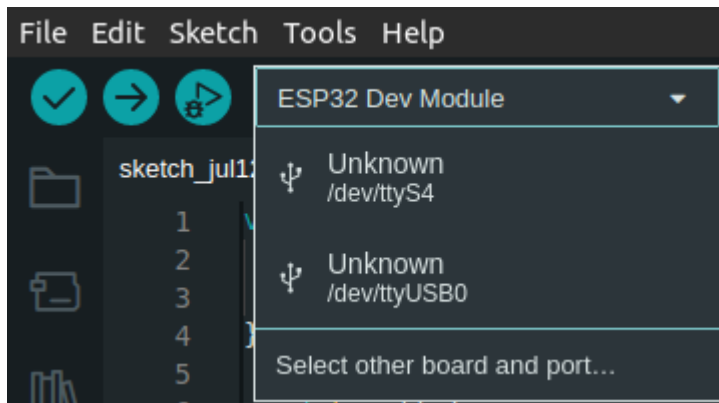


Cole separadamente cada URL separando cada uma com um comando da tecla ENTER, depois clique em “OK” nas duas guias para salvar e fechar.

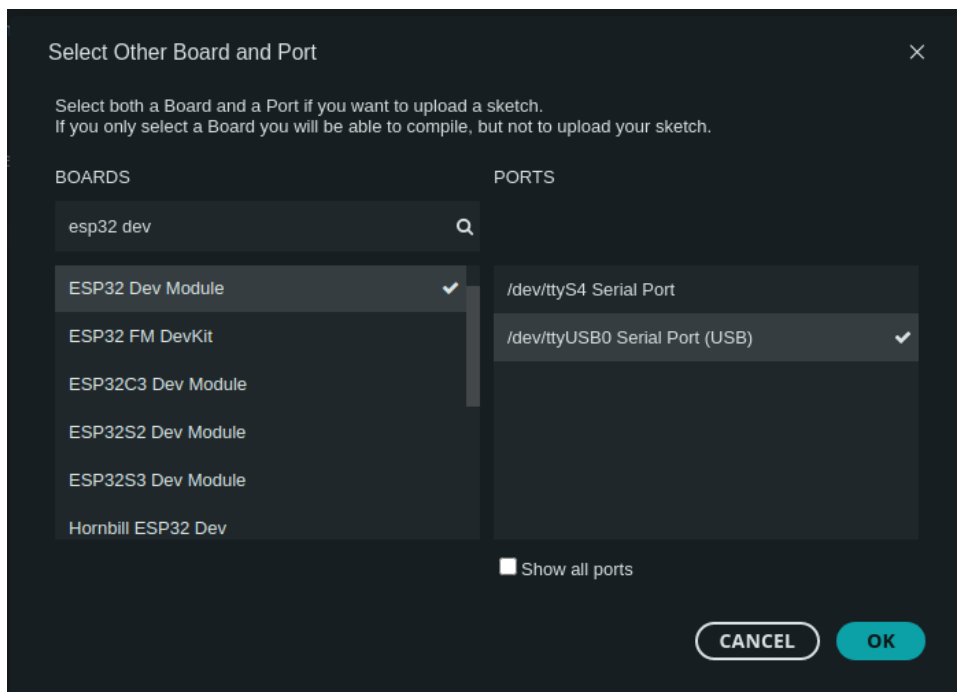
Integração da Arduino IDE com ESP32

Com os pacotes de dados instalados, conecte o ESP32 ao computador, para que possa ser reconhecido o modelo da placa e a porta USB correspondente para transmissão de dados.

Veja que ao conectar o ESP32 ele já começa a transmitir dados para a IDE que o reconhece em uma porta USB específica, conforme mostrado na imagem abaixo:



Em determinadas situações a detecção pode ser automática, caso não seja, selecione a opção apresentada acima **“Select other board and port”** para que selecione o modelo da placa do ESP32 que você está utilizando e qual a porta USB que ele está conectado. Clique em “OK” para salvar as configurações e fechar a guia.



Assim sua placa ESP32 já está se comunicando com sua Arduino IDE.

Bibliotecas dos Sensores

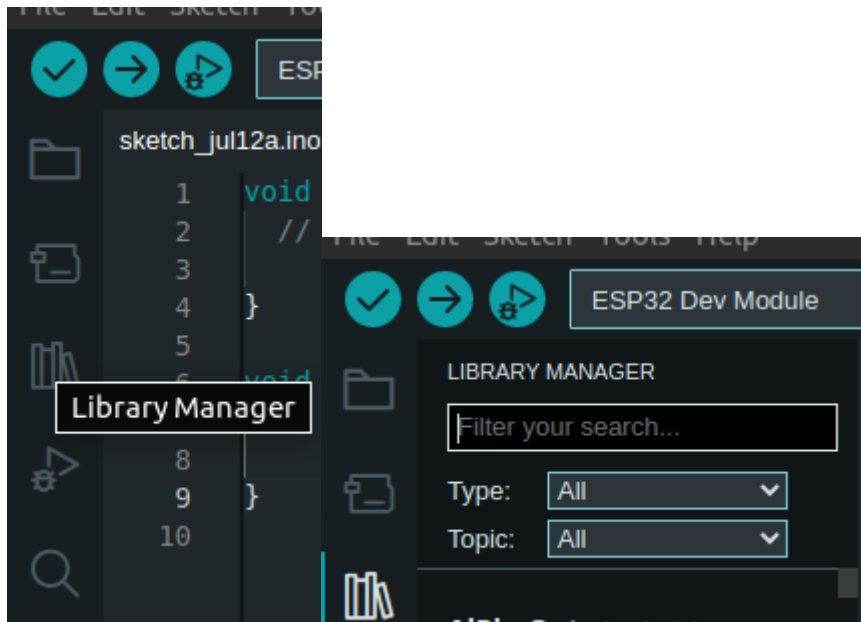
Basicamente com as configurações apresentadas acima, a programação do ESP32 pode ser feita e importada tranquilamente para a placa através da Arduino IDE. Existe também os sensores, que por sua vez também necessitam de uma configuração específica, importando as chamadas bibliotecas que são trechos de códigos que permitem que certas funções e/ou especificações sejam feitas pelos sensores e que as mesmas possam ser aproveitadas no código. Não utilizar estas bibliotecas pode (para não dizer que irá) causar inconsistências no código quando o mesmo for exportado para o ESP32. A explicação será mais aprofundada no capítulo de [Discussão](#) que irá abordar alguns erros apresentados pelos sensores.

Para se importar uma biblioteca há 2 maneiras:

Opção 1 – Na Arduino IDE vá em Sketch > Include Library > Manage Libraries para abrir o gerenciador de bibliotecas

Opção 2 – Acione as teclas de atalho **Ctrl + Shift + I** para abrir o gerenciador de bibliotecas

Opção 3 – Clique no menu lateral da Arduino IDE sobre o ícone de bibliotecas conforme a imagem abaixo:



Feito isto, basta utilizar a barra de pesquisa para localizar a sua biblioteca específica para o seu projeto e/ou pesquisar pelo nome do sensor que irá utilizar e ver qual mais lhe agrada.

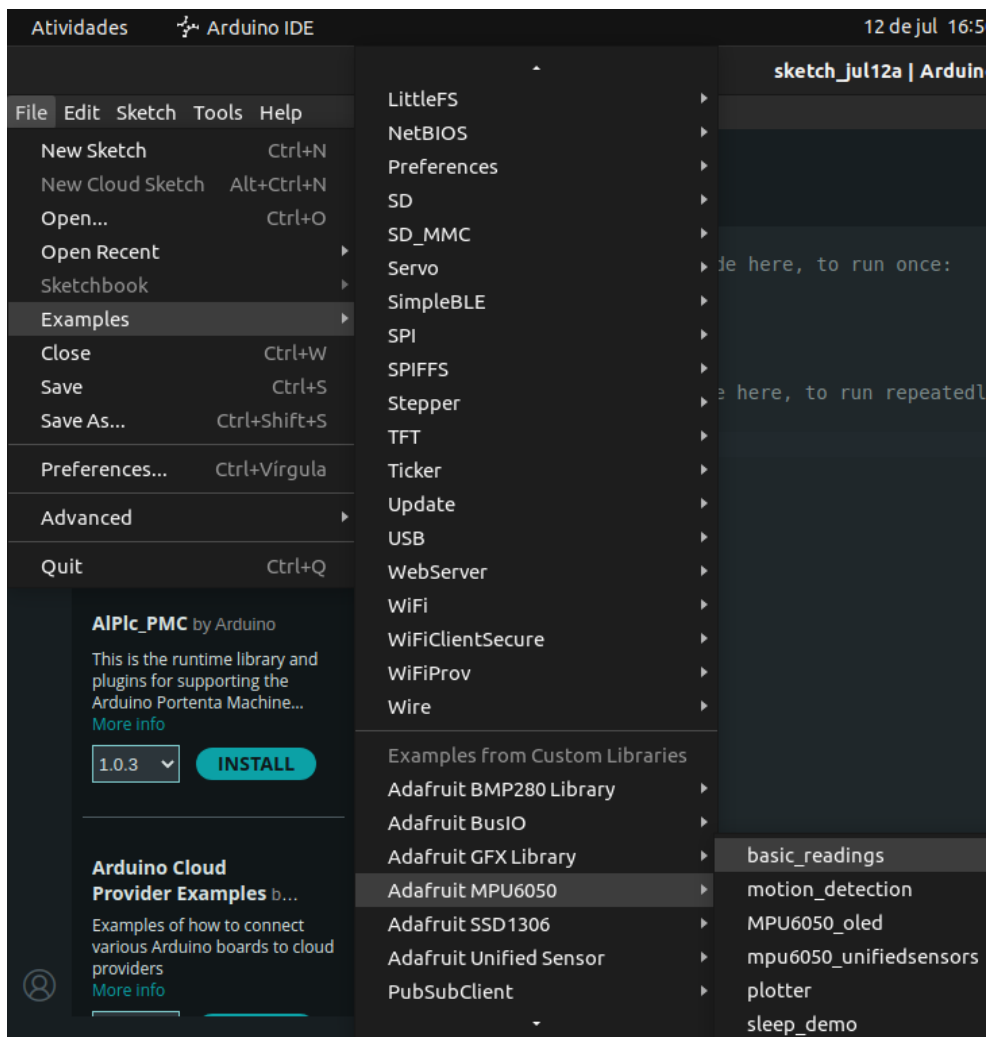
Programação

Com o ESP32 se comunicando com a Arduino IDE, que também já está configurada com as bibliotecas dos sensores que irá utilizar, agora você pode programar tranquilamente seus sensores, módulos e demais componentes sem nenhum problema, que ao ser exportado, caso não haja erros no código, será gravado no ESP32 executará seu código da maneira que foi descrita.

Com o ambiente pronto para testes (e como dito em capítulos anteriores), existem na Arduino IDE alguns códigos prontos para uso de testes de funcionalidade de cada sensor, veja a seguir como usar cada um.

Programação do Sensor MPU6050

Com as bibliotecas do sensor MPU6050 instaladas previamente, conseguimos através da Arduino IDE pegar um código base para o uso do sensor. Para que seja possível a biblioteca **Adafruit MPU6050** deve ser instalada conforme dito anteriormente. Com a biblioteca instalada, siga para o caminho especificado pela imagem abaixo:



Selecionando a opção de apresentada para gerar um código básico da leitura do sensor. Este código servirá de base para que seu projeto possa ser desenvolvido, programado e adaptado da melhor maneira possível.

A primeira parte do código que será exibida são as bibliotecas que o sensor utiliza e são necessárias para seu funcionamento:

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
```

Abaixo desta parte do código, será feita uma inicialização do sensor de forma que o mesmo possa ser utilizado livremente no seu código com todos os atributos disponíveis.

```
Adafruit_MPU6050 mpu;
```

Partindo então para a configuração do sensor para ter seu melhor funcionamento, começando com fazer uma leitura e uma confirmação para sabe se o sensor está conectado e pronto para uso da comunicação serial.

```

void setup(void) {
  Serial.begin(115200);
  while (!Serial)
    delay(10); // will pause Zero, Leonardo, etc until serial console opens

  Serial.println("Adafruit MPU6050 test!");

  // Try to initialize!
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
  Serial.println("MPU6050 Found!");
}

```

Abaixo inserimos as configurações para uso dos parâmetros do acelerômetro.

```

mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
Serial.print("Accelerometer range set to: ");
switch (mpu.getAccelerometerRange()) {
case MPU6050_RANGE_2_G:
  Serial.println("+2G");
  break;
case MPU6050_RANGE_4_G:
  Serial.println("+4G");
  break;
case MPU6050_RANGE_8_G:
  Serial.println("+8G");
  break;
case MPU6050_RANGE_16_G:
  Serial.println("+16G");
  break;
}

```

Depois inserimos as configurações para uso do giroscópio.

```

mpu.setGyroRange(MPU6050_RANGE_500_DEG);
Serial.print("Gyro range set to: ");
switch (mpu.getGyroRange()) {
case MPU6050_RANGE_250_DEG:
  Serial.println("+250 deg/s");
  break;
case MPU6050_RANGE_500_DEG:
  Serial.println("+500 deg/s");
  break;
case MPU6050_RANGE_1000_DEG:
  Serial.println("+1000 deg/s");
  break;
case MPU6050_RANGE_2000_DEG:
  Serial.println("+2000 deg/s");
  break;
}

```

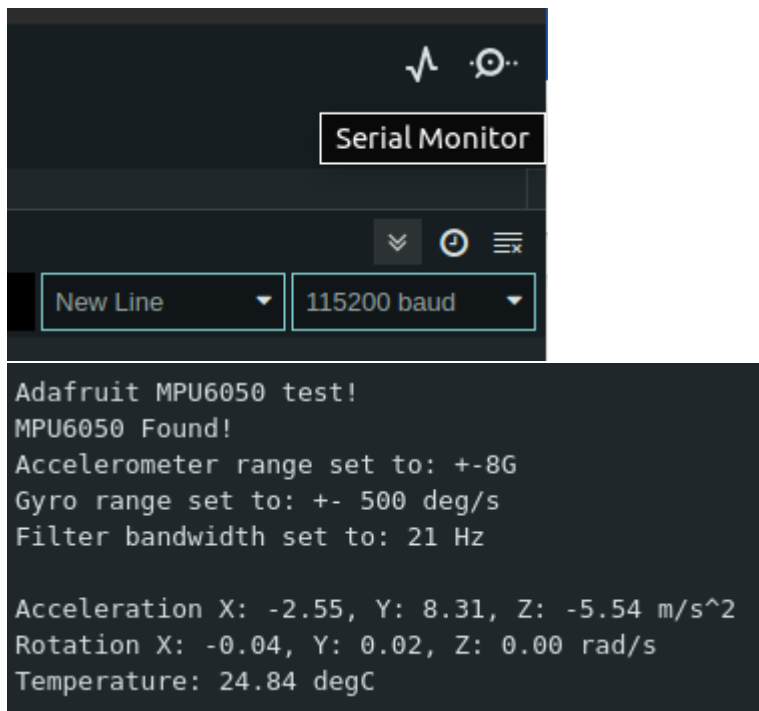
Finalizamos com um filtro de banda.

```
mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);  
Serial.print("Filter bandwidth set to: ");  
switch (mpu.getFilterBandwidth()) {  
case MPU6050_BAND_260_HZ:  
    Serial.println("260 Hz");  
    break;  
case MPU6050_BAND_184_HZ:  
    Serial.println("184 Hz");  
    break;  
case MPU6050_BAND_94_HZ:  
    Serial.println("94 Hz");  
    break;  
case MPU6050_BAND_44_HZ:  
    Serial.println("44 Hz");  
    break;  
case MPU6050_BAND_21_HZ:  
    Serial.println("21 Hz");  
    break;  
case MPU6050_BAND_10_HZ:  
    Serial.println("10 Hz");  
    break;  
case MPU6050_BAND_5_HZ:  
    Serial.println("5 Hz");  
    break;  
}
```

Por fim, selecionamos os parâmetros de leitura que utilizaremos e que foram informados na parte anterior do código através do **evento de sensores** criando variáveis de referência, para que assim as mesmas possam exibir os valores de leitura do sensor.

```
void loop() {  
  
  /* Get new sensor events with the readings */  
  sensors_event_t a, g, temp;  
  mpu.getEvent(&a, &g, &temp);  
  
  /* Print out the values */  
  Serial.print("Acceleration X: ");  
  Serial.print(a.acceleration.x);  
  Serial.print(", Y: ");  
  Serial.print(a.acceleration.y);  
  Serial.print(", Z: ");  
  Serial.print(a.acceleration.z);  
  Serial.println(" m/s^2");  
  
  Serial.print("Rotation X: ");  
  Serial.print(g.gyro.x);  
  Serial.print(", Y: ");  
  Serial.print(g.gyro.y);  
  Serial.print(", Z: ");  
  Serial.print(g.gyro.z);  
  Serial.println(" rad/s");  
  
  Serial.print("Temperature: ");  
  Serial.print(temp.temperature);  
  Serial.println(" degC");  
  
  Serial.println("");  
  delay(500);  
}
```

A programação concluída, temos o seguinte resultado no monitor serial.

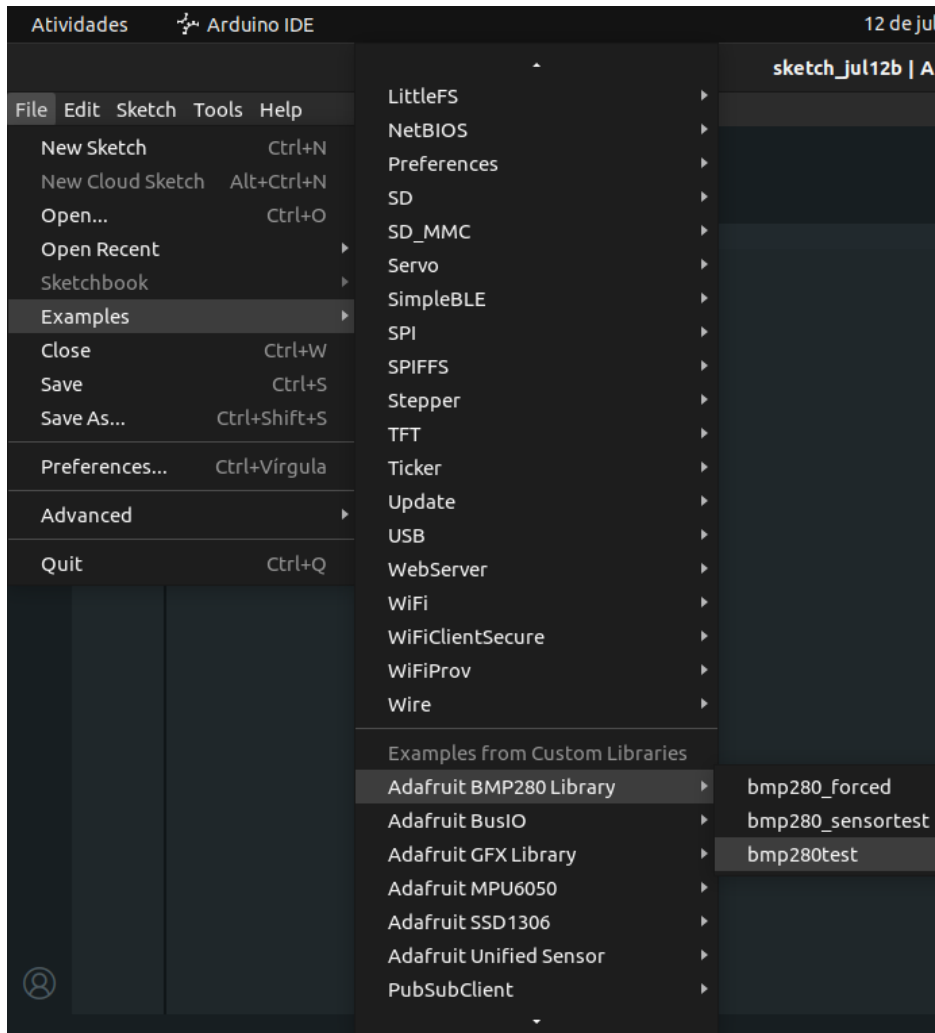


The screenshot shows the Serial Monitor interface with the title 'Serial Monitor'. Below the title bar, there are icons for a waveform and a refresh button. The main area displays the following text:

```
Adafruit MPU6050 test!  
MPU6050 Found!  
Accelerometer range set to: +-8G  
Gyro range set to: +- 500 deg/s  
Filter bandwidth set to: 21 Hz  
  
Acceleration X: -2.55, Y: 8.31, Z: -5.54 m/s^2  
Rotation X: -0.04, Y: 0.02, Z: 0.00 rad/s  
Temperature: 24.84 degC
```


Programação do Sensor BMP280

Com as bibliotecas do sensor **BMP280** instaladas previamente, conseguimos através da Arduino IDE pegar um código base para o uso do sensor. Para que seja possível a biblioteca **Adafruit BMP280 Library** e suas dependências devem ser instaladas conforme dito anteriormente. Com a biblioteca instalada, siga para o caminho especificado pela imagem abaixo:



Selecionando a opção de apresentada para gerar um código básico da leitura do sensor. Este código servirá de base para que seu projeto possa ser desenvolvido, programado e adaptado da melhor maneira possível.

A primeira parte do código que será exibida são as bibliotecas que o sensor utiliza e são necessárias para seu funcionamento:

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_BMP280.h>
```

Abaixo desta parte do código, será feita uma inicialização do sensor de forma que o mesmo possa ser utilizado livremente no seu código com todos os atributos disponíveis.

```
#define BMP_SCK  (13)
#define BMP_MISO (12)
#define BMP_MOSI (11)
#define BMP_CS   (10)

Adafruit_BMP280 bmp; // I2C
//Adafruit_BMP280 bmp(BMP_CS); // hardware SPI
//Adafruit_BMP280 bmp(BMP_CS, BMP_MOSI, BMP_MISO, BMP_SCK);
```

No caso deste sensor, ele pode se comunicar de 2 formas diferentes, analise seu projeto e leia o datasheet do seu projeto para saber se irá utilizar a configuração via **I2C** ou **SPI**, ambas permitem o funcionamento do sensor, porém contém atributos diferentes.

Partindo então para a configuração do sensor para ter seu melhor funcionamento, começando com fazer uma leitura e uma confirmação para saber se o sensor está conectado e pronto para uso da comunicação serial.

```
void setup() {
  Serial.begin(9600);
  while ( !Serial ) delay(100); // wait for native usb
  Serial.println(F("BMP280 test"));
  unsigned status;
  //status = bmp.begin(BMP280_ADDRESS_ALT, BMP280_CHIPID);
  status = bmp.begin();
  if (!status) {
    Serial.println(F("Could not find a valid BMP280 sensor, check wiring or "
    "try a different address!"));
    Serial.print("SensorID was: 0x"); Serial.println(bmp.sensorID(),16);
    Serial.print("      ID of 0xFF probably means a bad address, a BMP 180 or BMP 085\n");
    Serial.print("      ID of 0x56-0x58 represents a BMP 280,\n");
    Serial.print("      ID of 0x60 represents a BME 280.\n");
    Serial.print("      ID of 0x61 represents a BME 680.\n");
    while (1) delay(10);
  }

  /* Default settings from datasheet. */
  bmp.setSampling(Adafruit_BMP280::MODE_NORMAL, /* Operating Mode. */
    Adafruit_BMP280::SAMPLING_X2, /* Temp. oversampling */
    Adafruit_BMP280::SAMPLING_X16, /* Pressure oversampling */
    Adafruit_BMP280::FILTER_X16, /* Filtering. */
    Adafruit_BMP280::STANDBY_MS_500); /* Standby time. */
}
```

No caso deste sensor também há questões mais técnicas que a código de teste informa, são configurações técnicas, caso necessário, consulte o datasheet do sensor para melhor entendimento.

Com as configurações dos atributos do sensor concluídas, podemos finalizar a programação focando na exibição dos dados.

```

void loop() {
  Serial.print(F("Temperature = "));
  Serial.print(bmp.readTemperature());
  Serial.println(" *C");

  Serial.print(F("Pressure = "));
  Serial.print(bmp.readPressure());
  Serial.println(" Pa");

  Serial.print(F("Approx altitude = "));
  Serial.print(bmp.readAltitude(1013.25)); /* Adjusted to local forecast! */
  Serial.println(" m");

  Serial.println();
  delay(2000);
}

```

A programação concluída, temos o seguinte resultado.

[HOVE COMPLICAÇÕES NO FUNCIONAM, ENTO DO SENSOR QUE NÃO PERMITIRAM A VISUALIZAÇÃO CORRETA DOS DSDOS]

Programação do Sensor ACS712

[HÁ REFERÊNCIAS ESCASSAS PARA O USO DESTE SENSOR NAS PESQUISAS REALIZADAS, PORTANTO NÃO HÁ COMO APRESENTAR O FUNCIONAMENTO E PROCESSO PARA O USO DESTE SENSOR]

[HOVE COMPLICAÇÕES NO FUNCIONAM, ENTO DO SENSOR QUE NÃO PERMITIRAM A VISUALIZAÇÃO CORRETA DOS DADOS]

Programação do Módulo de Cartão Micro SD

Com as bibliotecas do Módulo de cartão SD instaladas previamente, conseguimos através da Arduino IDE pegar um código base para o uso do sensor. Diferente dos sensores anteriores, não é necessário instalar uma biblioteca para uso do módulo de cartão SD. Siga para o caminho especificado pelo caminho abaixo para acessar o código de leitura e escrita de arquivos:

Files > Examples > SD > ReadWrite

Lembre-se que será criado um arquivo

Selecionando a opção de apresentada para gerar um código básico da leitura do sensor. Este código servirá de base para que seu projeto possa ser desenvolvido, programado e adaptado da melhor maneira possível.

A primeira parte do código que será exibida são as bibliotecas que o módulo utiliza e são necessárias para seu funcionamento:

```

#include <SPI.h>
#include <SD.h>

```

Abaixo desta parte do código, será feita uma inicialização do módulo de forma que o mesmo possa ser utilizado livremente no seu código com todos os atributos disponíveis, este procedimento será feito inicializando um **arquivo de gravação**. Também será exibida uma configuração para caminho de transmissão de dados através de uma conexão específica.

```
File myFile;
const int CS = 5;
```

Partindo então para a configuração do sensor para ter seu melhor funcionamento, começando com fazer uma leitura e uma confirmação para saber se o sensor está conectado e pronto para uso da comunicação serial.

```
void setup() {
  Serial.begin(9600);    // Set serial baud rate to 9600
  delay(500);
  while (!Serial) { ; } // wait for serial port to connect. Needed for native USB port only
  Serial.println("Initializing SD card...");
  if (!SD.begin(CS)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

  WriteFile("/test.txt", "ORBISAT");
  ReadFile("/test.txt");
}
```

Abaixo da inicialização também é inserido um comando para abrir um arquivo denominado *“test.txt”* localizado na raiz do sistema, ou início da pasta, onde neste mesmo arquivo será feito a gravação da informação de texto *“ORBISAT”*. Logo em seguida é apresentado um comando para ler as informações contidas neste arquivo *“test.txt”* que foi criado e preenchido no comando anterior.

Para que estas 2 funções sejam executadas de maneira correta é necessário criá-las de maneira independente como funções de código. Abaixo está o desenvolvimento das 2 funções, que estão fora do escopo do **void loop()** e **void setup()**.

Código de **Criação e Gravação** em um Arquivo .txt

```
void WriteFile(const char * path, const char * message){
  // open the file. note that only one file can be open at a time,
  // so you have to close this one before opening another.
  myFile = SD.open(path, FILE_WRITE);
  // if the file opened okay, write to it:
  if (myFile) {
    Serial.printf("Writing to %s ", path);
    myFile.println(message);
    myFile.close(); // close the file:
    Serial.println("completed.");
  }
  // if the file didn't open, print an error:
  else {
    Serial.println("error opening file ");
    Serial.println(path);
  }
}
```

Código de **Leitura** de um Arquivo .txt

```
void ReadFile(const char * path){
  // open the file for reading:
  myFile = SD.open(path);
  if (myFile) {
    Serial.printf("Reading file from %s\n", path);
    // read from the file until there's nothing else in it:
    while (myFile.available()) {
      Serial.write(myFile.read());
    }
    myFile.close(); // close the file:
  }
  else {
    // if the file didn't open, print an error:
    Serial.println("error opening test.txt");
  }
}
```

A programação concluída, temos o seguinte resultado apresentado no Monitor serial.

```
Initializing SD card...
initialization done.
Writing to /test.txt completed.
Reading file from /test.txt
ORBISAT
```

Repare que a última linha é exatamente a mesma informação de texto passada na função de escrita de arquivo.

Discussão

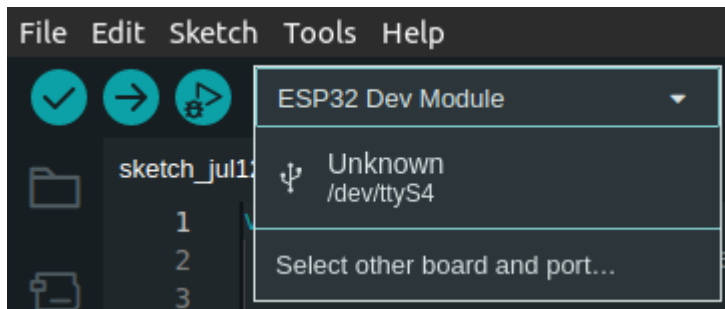
Este capítulo tem objetivo de apresentar observações para o melhor funcionamento do ESP32, sensores, bibliotecas e demais. A apresentação de erros ou problemas ajudará a lidar com inconsistências durante a produção do projeto final.

Problemas com ESP32

Como apresentado durante este relatório é de extrema importância pesquisar sobre o modelo do ESP32 que se está utilizando, pois cada modelo pode conter um pacote de dados diferente, causando inconsistências e possíveis problemas na execução de códigos. Portanto sempre consulte o site da fabricante tanto da placa de microcontrolador quanto dos sensores para melhor entendimento e integração um com o outro.

Problemas com Porta USB

Em determinadas ocasiões, ao conectar o EPS32 em uma porta USB, ele pode não ser reconhecido, como a figura abaixo mostra.



Algumas soluções podem ser feitas quando a isto:

1. Problemas com a porta USB: em algumas situações a porta USB do computador pode estar danificada, neste caso, troque de porta USB;
2. Utilizar cabo de energização: muitos cabos USB comprados separadamente do produto, tendem a ser cabos de energia apenas, ou seja, apenas carregam ou transmitem energia, mas não transmite dados, nesta situação, troque o cabo. Há também a possibilidade de o cabo ser um cabo de dado, mas não está conectando ao ESP32, neste caso verifique se:
 - a. O ESP32 acendeu o LED: problema no cabo, não está transmitindo dados, troque de cabo;
 - b. O ESP32 **NÃO** acendeu o LED: o cabo USB pode estar com fios internos quebrados ou o ESP32 pode estar queimado;

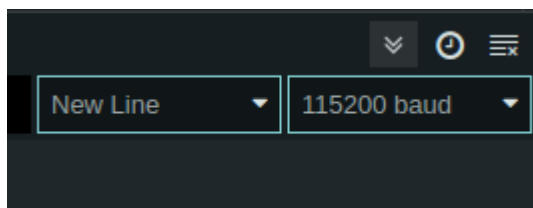
Problemas com MPU6020

Durante os testes realizados com este sensor não foram identificados erros que ocasionassem no não funcionamento ou não visualização dos dados, para que seja possível utilizar este sensor corretamente é necessário seguir o código apresentado anteriormente e instalar a biblioteca **Adafruit MPU6050** através do gerenciador de bibliotecas.

Caso a IDE informe que tenha versões atualizadas, verifique se a atualização não irá prejudicar o funcionamento do seu projeto. Caso solicite, instale as demais dependências junto da biblioteca quando ela estiver sendo instalada.

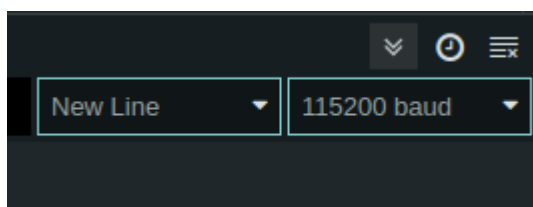


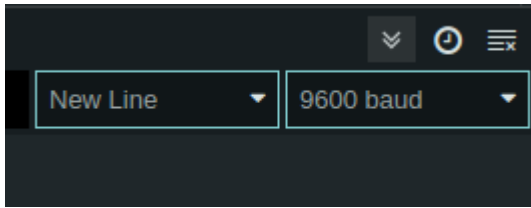
Para visualização dos dados, no monitor serial selecione a opção apresentada abaixo, caso contrário os dados serão apresentados de maneira incorreta ou desconfiguradas.



Problemas com BMP280

[HOVE COMPLICAÇÕES NO FUNCIONAM, ENTO DO SENSOR QUE NÃO PERMITIRAM A VISUALIZAÇÃO CORRETA DOS DADOS]





```
Could not find a valid BMP280 sensor, check wiring or try a different address!
SensorID was: 0x0
  ID of 0xFF probably means a bad address, a BMP 180 or BMP 085
  ID of 0x56-0x58 represents a BMP 280,
  ID of 0x60 represents a BME 280.
  ID of 0x61 represents a BME 680.
```

Problemas com ACS712

[HOUVE COMPLICAÇÕES NO FUNCIONAM, ENTO DO SENSOR QUE NÃO PERMITIRAM A VISUALIZAÇÃO CORRETA DOS DADOS]

Problemas com Módulo de Cartão Micro SD

Seguindo os passos de configuração do Módulo de Cartão SD apresentados anteriormente, não tende a haver nenhum erro, porém alguns podem aparecer não por causa da programação em si, mas por configurações específicas a respeito do **cartão micro SD**.

```
Initializing SD card...
initialization failed!
```

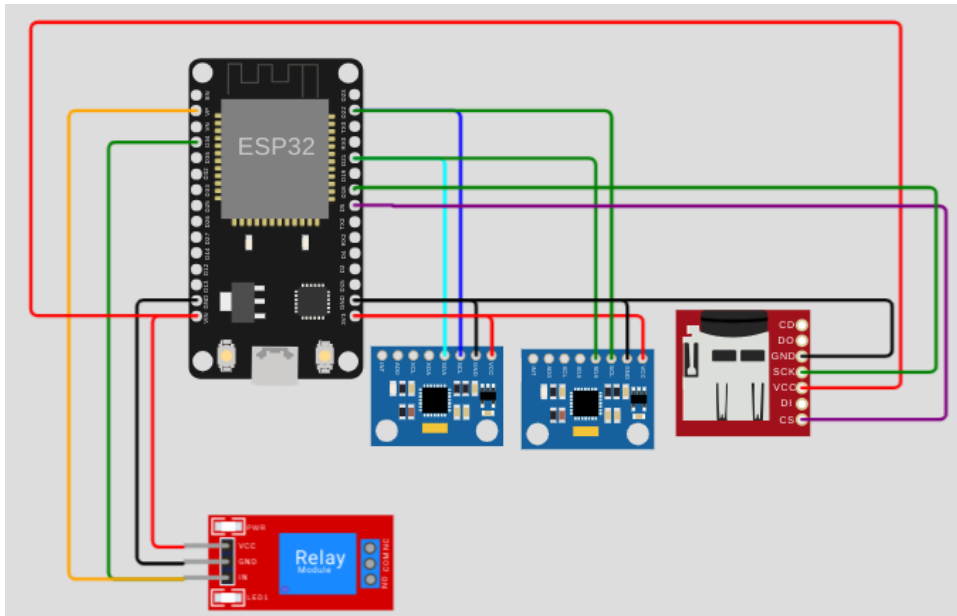
A biblioteca da Arduino IDE oferece suporte aos sistemas de arquivos FAT16 e FAT32, dependendo da formatação do seu cartão SD ele pode não ser reconhecido ou pode ficar impossibilitado de efetuar a gravação de informações no arquivo.

Certifique-se de que seu cartão SD esteja formatado apenas com esses dois tipos, caso contrário, ocorrerá um erro de inicialização como apresentado acima. Em alguns casos pode ocorrer de uma atualização suportar outros formatos, mas é sempre bom verificar a biblioteca que estiver utilizando.

Conclusão

Em linhas gerais, o projeto final terá um planejamento parecido com a figura abaixo. Os códigos acima serão adaptados para transmitirem os dados e informações com nossa própria formatação, além de serem transpostos para um formato JSON, requisito essencial solicitado no edital do projeto.

A imagem abaixo não representa totalmente a aparência final do projeto, pois o simulador não contém alguns sensores utilizados e a pinagem de conexão de alguns sensores, estão em modelos diferentes dos utilizados no projeto final, servindo apenas como guia e referência de montagem e organização.



Recomendações

Sempre tenha em mente que hardware funciona especificamente conforme ele é planejado, diferente de um código que através de diversas maneiras distintas se obtém o mesmo resultado. Sempre que possível projete suas ideias com os datasheets de cada sensor e de cada placa com microcontrolador lado a lado, para assim, entender a melhor maneira de cada um se encaixar e apresentarem juntos o melhor resultado.

Referências

Arduinoecia. **Usando o BMP280 com Arduino Uno**. Disponível em: <https://www.arduinoecia.com.br/bmp280-pressao-temperatura-altitude/>. Acesso em: 06 jul.2023

Mundoprojetado. **Acelerômetro MPU6050 com o Arduino**. Disponível em: <https://mundoprojetado.com.br/acelerometro-mpu6050-arduino/>. Acesso em: 12 jun.2023

Wikipédia, a enciclopédia livre. **Barômetro**. Disponível em: <https://pt.wikipedia.org/wiki/Bar%C3%B3metro>. Acesso em: 12 jun.2023

Robocore. **Sensor De Pressão e Temperatura BMP280**. Disponível em: <https://www.robocore.net/sensor-ambiente/sensor-de-pressao-e-temperatura-bmp280>. Acesso em: 12 jun.2023

L GUSTAVO Exercícios e Projetos do Livro. **Projeto 35 - Sensor de Corrente ACS712 com ESP32-DOIT - Projetos Jetsons**. Disponível em: [Projeto 35 - Sensor de Corrente ACS712 com ESP32-DOIT - Projetos Jetsons](#). Acesso em: 07 jun.2023

EngEasier | Automação com Arduino e ESP32. **COMPUTADOR NÃO RECONHECE A ESP32? Como resolver**. Disponível em: [COMPUTADOR NÃO RECONHECE A ESP32? Como resolver](#). Acesso em: 05 jun.2023

BINARYUPDATES. **How to Setup and Program ESP32 Microcontroller– Complete Guide**. Disponível em: [How to Setup and Program ESP32 Microcontroller– Complete Guide](#). Acesso em: 05 jun.2023

Wokwi. **Wokwi Simulate IoT Projects in Your Browser**. Disponível em: <https://wokwi.com/>. Acesso em: 04 jun.2023

Electronicwings. **MPU6050 Gyroscope Interfacing with ESP32**. Disponível em: <https://www.electronicwings.com/esp32/mpu6050-gyroscope-interfacing-with-esp32>. Acesso em: 03 jun.2023

Electronicwings. **BMP280 Barometer Sensor Interfacing with ESP32**. Disponível em: <https://www.electronicwings.com/esp32/bmp280-barometer-sensor-interfacing-with-esp32>. Acesso em: 03 jun.2023

Randomnerdtutorials. **ESP32 with MPU-6050 Accelerometer, Gyroscope and Temperature Sensor (Arduino)**. Disponível em: <https://randomnerdtutorials.com/esp32-mpu-6050-accelerometer-gyroscope-arduino/>. Acesso em: 03 jun.2023

Esp32learning. **ESP32 and BMP280 sensor example**. Disponível em: <http://www.esp32learning.com/code/esp32-and-bmp280-sensor-example.php>. Acesso em: 03 jun.2023

Electronicwings. **MicroSD Card Interfacing with ESP32**. Disponível em: <https://www.electronicwings.com/esp32/microsd-card-interfacing-with-esp32>. Acesso em: 02 jun.2023

Fernando K Tecnologia. **Como gravar arquivos em microSD com ESP32**. Disponível em: <https://www.fernandok.com/2018/11/como-gravar-arquivos-em-microsd-com.html>. Acesso em: 02 jun.2023

Robocore. **Instalando o Driver do NodeMCU e ESP32**. Disponível em: <https://www.robocore.net/tutoriais/instalando-driver-do-nodemcu>. Acesso em: 02 jun.2023

Randomnerdtutorials. **ESP32 with MPU-6050 Accelerometer, Gyroscope and Temperature Sensor (Arduino)**. Disponível em: <https://randomnerdtutorials.com/esp32-mpu-6050-accelerometer-gyroscope-arduino/>. Acesso em: 02 jun.2023

Espressif. **E ESP32-WROOM-32D & ESP32-WROOM-32U Datasheet**. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf. Acesso em: 28 jun.2023

Espressif. **ESP32-WROOM Datasheet**. Disponível em: <https://www.espressif.com/en/support/documents/certificates?keys=ESP32-WROOM-32D>. Acesso em: 28 jun.2023

Internet e Coisas. **ESP32 ESP8266 Armazenando Configuração com JSON e SPIFFS - leC103**. Disponível em: [ESP32 ESP8266 Armazenando Configuração com JSON e SPIFFS - leC103](#). Acesso em: 15 mai.2023

Circuits4you. **ESP32 DevKit ESP32-WROOM GPIO Pinout**. Disponível em: <https://circuits4you.com/2018/12/31/esp32-devkit-esp32-wroom-gpio-pinout/>. Acesso em: 09 mai.2023

PET - Tecnologia em Eletrônica e Computação. **Tutorial - Enviando um JSON com ESP32**. Disponível em: <https://pettec.unifei.edu.br/wp-content/uploads/2021/01/Tutorial-Enviando-um-JSON-com-ESP32.pdf>. Acesso em: 16 abr.2023