**Guia**

**de Referência**

**Manoel Messias**

Sumário

| **Capítulo 1: Ubuntu …………………………….………………………………………….** | 6 |
| --- | --- |
| ***Instalação do Ubuntu 24.04 no Raspberry Pi 4* ……………………………………** | 7 |
| *Método 1: Através da imagem* **………………………………………………………..** | 7 |
| *Método 2: Usando a ferramenta Raspberry Pi Imager* **…………………………….** | 8 |
| ***Configurando o Ubuntu* ………………………………………………………………..** | 9 |
| *Configuração inicial* **………………………………………………………………….…** | 9 |
| *Instalando o ssh* **………………………………………………………………………..** | 9 |
| ***Comandos do Ubuntu* ………………………………………………………….………** | 11 |
| *Diretório e arquivos* **……………………………………………………**.**………………** | 11 |
| *Sudo* **………….…………………..……………………………………………………..** | 11 |
| *Usuário root* **……………………………………………….…………………………….** | 11 |
| ***Instalação de aplicações* ……………………………….……………………………...** | 13 |
| *1. Pacotes Debian* **…………………………….………………………………………..** | 13 |
| *2. Pacotes .tar.xz* **……………………………………………………………………….** | 15 |
| *Instalando o FTP* **……………………………………………………………………….** | 15 |
| ***Serviços* ……………………………..…..……………………………………………….** | 17 |
| *Criação* **………………………….……………………………………………………….** | 17 |
| *Comandos* **……......…………….……………………………………………………….** | 18 |
| *Executando scripts* **.………….………………..………………………………………..** | 19 |
| ***Python* ………………………….………………………………………………………….** | 21 |
| *threading* **.……………………..………………..………………………………………..** | 21 |
| *rich* **.……….………………….…………..……..………………………………………..** | 21 |
| *pySerial* **.…………….……………..…………………………………………..………...** | 102 |
| *subprocess* **.…………………………..………………………………………………….** | 102 |
| *socket* **.……..…………………………………..………………………………………..** | 102 |
| ***Github* ………………….………………………………………..………………………...** | 102 |
|  |  |
|  |  |
|  |  |
| **Capítulo 2: O Raspberry …………………………………………………….……………** | 102 |
| ***GPIOs* ………………………………………………………………………………….…..** | 102 |
| *Instalando GPIOs* **.…………………………..………………………………………….** | 102 |
| *Saídas digitais* **.……………………….....……………………….………………….….** | 102 |
| *Entradas digitais* **.…………………………..…………………………..……………….** | 102 |
| ***UART* ………………………………………………………………………………….…...** | 102 |
| *O que é UART* **.………..…………………..…………………………..…………….….** | 102 |
| *Configurando a UART* **.……………………………………………....…………….….** | 102 |
| *Tornando-a acessível* **.……………………..……..…………………..…………….….** | 102 |
| *Usando a UART* **.……………………..……..…………………..………………….….** | 102 |
| ***LCD* …………………………………………………………………………………..…….** | 102 |
| *Conexão Física* **.…………………………..…………………………..….…………….** | 102 |
| *Bibliotecas* **.………………………………..…………………………..….…………….** | 102 |
| *Driver de controle* **.………………………..…………………………..….…………….** | 102 |
| *Funções básicas* **.…………………………..…………………………..…………..….** | 102 |
| ***Câmera* ……………………………………………………………………………….…..** | 102 |
|  |  |
| **Capítulo 3: O ROS …………………………………………………………………………** | 102 |
| ***Introdução* ………………………………………………………………………………..** | 102 |
| ***Instalação* ………………………………………………………………………………...** | 102 |
| ***Packages e Node* ………………………………………………………………………...** | 102 |
|  |  |
| **Capítulo 4: Microcontroladores ………………………………………………………...** | 102 |
| ***Arduino* …………………………………………………………………………………...** | 102 |
| *Introdução* **.…………………………..…………………………..…………………..….** | 102 |
| *GPIOs* **.…………………………..…………………………..……………………….….** | 102 |
| *PWM* **.…………………………..…………………………..……………………….……** | 102 |
| *Timers* **.………………………..…………………………..…………………….….……** | 102 |
| *Interrupções externas* **.…………..………….……………..………………….….……** | 102 |
| *Pin Change Interrupt* **.…….……..………….……..……..………………….….……** | 102 |
| ***STM32* …………………………………………………………………………….……….** | 102 |
| *Introdução* **.…………………………..…………………………..…………………..….** | 102 |
| *Programando* **.……..…….…………..…………………………..…………………..….** | 102 |
| *GPIO’s* **.…………………………..…………………………..……………………….….** | 102 |
| *Encoder Interface* **…..………..…………………………..……………………….……** | 102 |
| *Leitura Analogica* **.…………………………..…………………..…………………..….** | 102 |
| *EXTI* **.………….………………………..…………………..………….……………..….** | 102 |
| ***Sensores* ………………..…………………………………………………………….…..** | 102 |
| *HC-SR04* **.…..………………...………………….………..………………………...….** | 102 |
| *TCRT5000* **.…..………...……..…………………..………..………………………..….** | 102 |
| *Encoder* **.…..………...……..…………………..………..…………………………..….** | 102 |
| *MPU6050* **.…..…….…………..…………………..………..………………………..….** | 102 |
| ***Atuadores* ...…………..…………………………………………...……...……………...** | 102 |
| *Ponte H e motores DC* **.…………………………..…………………..…………..……** | 102 |
| *Servo motor* **.…………………………..…………………..………….……………..….** | 102 |
|  |  |
| **Capítulo 5: Odometria ……………………………………………………………………** | 102 |
| ***Coletando dados* ………………………………………………………………………..** | 102 |
| *Velocidade linear* **.…………………..…………………..………….……………....….** | 102 |
| *Encoder* **.……………..…………………..………….……………....................….** | 102 |
| *Acelerômetro* **.……………..……………..………….……………....................….** | 102 |
| *Velocidade angular* **..………………..…………………..………….……………....….** | 102 |
| *Encoder* **.……………..…………………..………….……………....................….** | 102 |
| *Giroscópio* **.…………..…………………..………….……………....................….** | 102 |
| *Equações de posicionamento* **..………………..………………….……………....….** | 102 |
| ***Estimativa - Filtro de Kalman* ……………………………..…………………………..** | 102 |
| *Introdução* **………………………..…………………..………….…………………..….** | 102 |
| ***Controladores* …………………………………………………………………………...** | 102 |
|  |  |
| **Capítulo 6: Seguindo uma linha …………………………..…………………….………** | 102 |
|  |  |
|  |  |
| **REFERÊNCIAS …………………………………………………………………………...** | 102 |

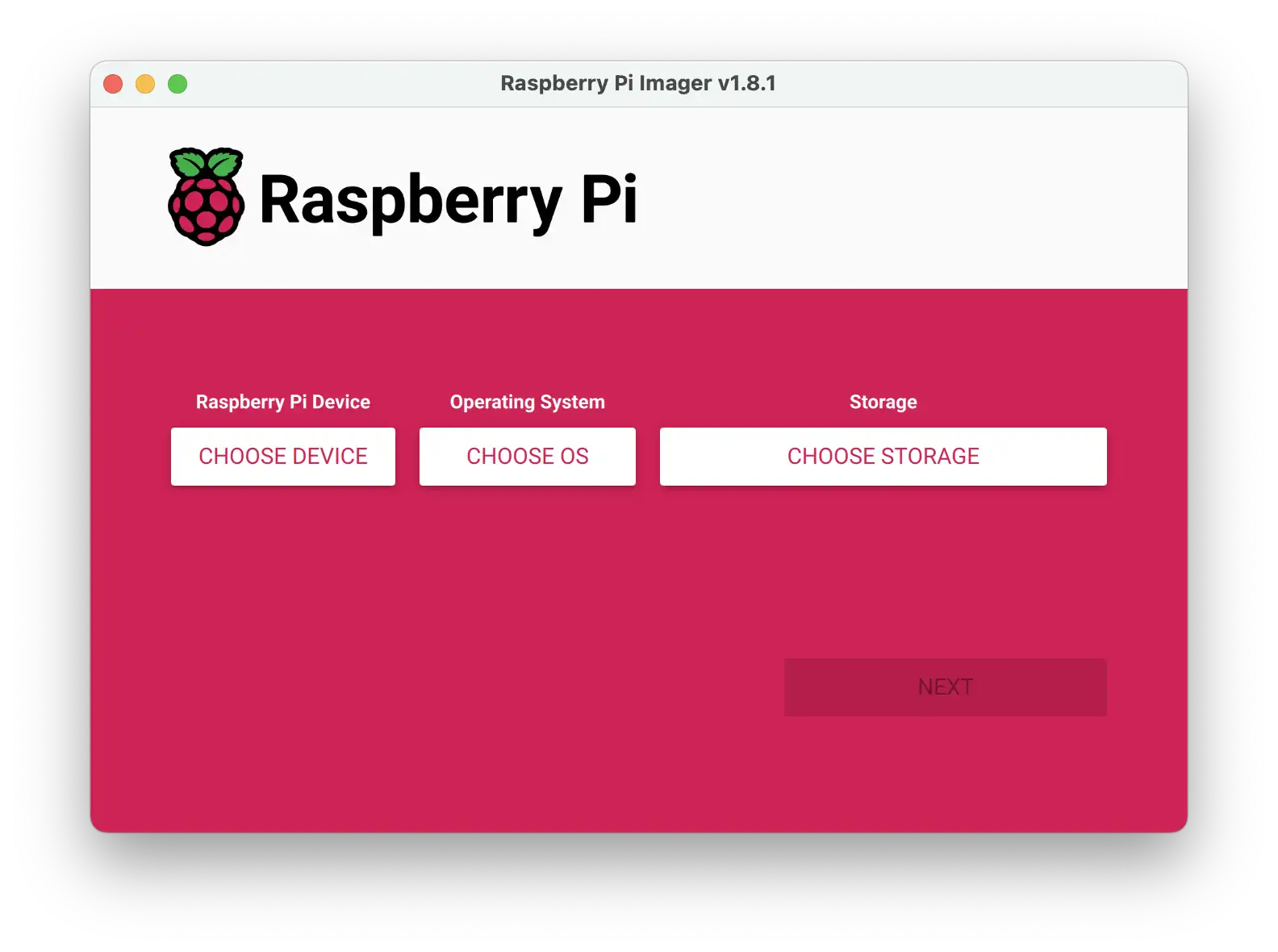
****

Instalação do Ubuntu 24.04 no Raspberry Pi 4

Para esse processo foi utilizado o Raspberry Pi 4 model B versão de 8gb de memória RAM e um cartão de memória de 64gb. Aqui será apresentado dois métodos de instalação do sistema operacional Ubuntu 24.04 LTS versão desktop. Para ambos os métodos será necessário que o cartão de memória seja retirado do Raspberry e conectado a um computador.

Para os dois métodos será necessário a instalação da ferramenta **Raspberry Pi Imager** que está disponível no endereço: <https://www.raspberrypi.com/software/>.

Esta é a ferramenta quando iniciada:



**Método 1: Através da imagem**

**1.1.** Baixando a imagem:

O arquivo da imagem pode ser baixado no link <https://cdimage.ubuntu.com/releases/noble/release/>, onde nesse tutorial foi escolhida a opção Preinstalled desktop image. Realize o download do arquivo compactado.

**1.2.** Instalando a Imagem:

1.2.1. Abra a ferramenta Raspberry Pi Imager. Selecione o dispositivo na opção CHOOSE DEVICE, aqui foi escolhido *Raspberry Pi 4.*

1.2.2. Na opção CHOOSE OS selecione “use custom” e escolha a imagem baixada.

1.2.3. Na opção CHOOSE STORAGE selecione o cartão SD.

1.2.4. Clique em next e o processo será iniciado.

**Método 2: Usando a ferramenta Raspberry Pi Imager**

**2.1.** Abra a ferramenta Raspberry Pi Imager. Selecione o dispositivo na opção CHOOSE DEVICE, aqui foi escolhido *Raspberry Pi 4.*

**2.2.** Na opção CHOOSE OS selecione “other general-purpose OS” em seguida escolha Ubuntu e depois a versão.

**2.3.** Instalando o Ubuntu server: Ao tentar instalar o ubuntu server ele irá já criar um usuário, você pode definir o nome do usuário e a senha clicando em **Edit Settings** quando ele aparecer. Também é possivel configurar uma rede wi-fi e ativar o ssh.

**2.4.** Na opção CHOOSE STORAGE selecione o cartão SD.

**2.5.** Clique em next e o processo será iniciado.

Configurando o Ubuntu

**1. Configuração inicial:**

Após ter concluído a instalação agora é hora de configurá-lo. Durante a primeira inicialização basta seguir o passo-a-passo apresentado na tela, adicionar um nome de usuário e uma senha. A senha é necessária para que você consiga dar comandos de super-usuário, este tópico será abordado mais à frente. Terminado o passo-a-passo seu ubuntu está pronto para ser usado normalmente. Para o **Ubuntu Server** a configuração é feita durante a instalação.

**2. Instalando o ssh:**

O SSH permite que você acesse o sistema de arquivos e até dê comandos no Raspberry através de outro computador na mesma rede.

Neste tipo de protocolo existe o cliente e o servidor. O cliente é aquele que está enviando os comandos e recebendo as respostas, e o servidor é aquele que recebe os comandos e envia uma resposta.

No Ubuntu o cliente SSH vem instalado por padrão, já o servidor SSH não vem. Para instalar ambos os recursos, cliente e servidor, basta abrir um terminal e executar o seguinte comando:

sudo apt install openssh-server openssh-client

A senha será solicitada, o motivo desta solicitação é explicada posteriormente.

*2.1. O endereço IP:*

Para usar o ssh é necessário primeiro conectar o seu Raspberry em uma rede e ter outro computador também conectado a esta mesma rede. O próximo passo é saber o endereço de IP, sendo este este uma sequência de 4 números que servirão para o outro computador encontrar seu Raspberry.

De forma geral e simplificada, podemos dizer que o formato do IP é 255.255.255.255 onde cada um desses 4 números podem variar entre 0 e 255. Para saber o IP dele, ou de qualquer outra máquina, basta abrir um terminal e digitar o seguinte comando:

hostname -I

Apenas o IP irá aparecer diretamente.

*2.2. Usando o ssh:*

Agora basta ir no outro computador que tem que ter o cliente SSH instalado (Ubuntu e windows a partir do 10 já possuem essa função por padrão), abrir um terminal (cmd no windows) e digitar o seguinte comando:

ssh usuario@ip

Onde “usuario” é o nome de usuário que você criou no processo de instalação e “ip” é um endereço de IP visto anteriormente.

O computador pedirá uma confirmação na primeira vez que você for se conectar, é necessário digitar ***yes*** e depois será necessário digitar a senha do usuário do Ubuntu (o que você está tentando se conectar - o servidor ssh). Após isso você já estará dentro de um terminal referente ao seu Raspberry.

**3. Conectando no wifi:**

Na versão desktop, o wifi pode ser ativado e conectado através da interface gráfica. Já na versão server você pode conectar-se a uma rede acessando o arquivo localizado na pasta /etc/netplan/50-cloud-init.yaml e adicionar as seguintes linhas abaixo de ‘***access-points***’:

Onde “*SSID*” é o nome da rede e “*senha*” é a senha da rede. Esse é o jeito mais fácil de fazer isso no ubuntu server. O arquivo deve ser aberto com permissão de superusuario (**sudo**).

Comandos do Ubuntu

O diferencial do Ubuntu, quando comparado a outros sistemas operacionais como o windows é a capacidade que temos de manipular recursos do sistema ou simples arquivos, nos dando o maior poder e controle do sistema como um todo. Isso é feito através dos diversos comandos que podem ser executados no terminal. Ao abrir o terminal seu diretório inicial é o diretório do usuário. Abaixo serão apresentados alguns comandos do Ubuntu:

**Diretório e arquivos:**

**1.** ls : Este comando, sozinho, lista os arquivos e pastas presentes no diretório atual.

1.1. ls part\* : **part** é a parte inicial do diretório ou arquivo que você quer listar. Ele exibirá todos os arquivos e diretórios presentes no diretório atual que se inicie com o valor dado.

1.2. ls -l file : **file** é o nome do arquivo, pasta ou caminho. Apresenta algumas informações do arquivo/pasta especificada.

**2.** cd caminho : Este comando altera o valor do diretório atual para o diretório referente ao **caminho**especificado.

**3.** mkdir nome : Cria uma pasta com o **nome** passado dentro do diretório atual.

**Sudo:**

O comando sudo permite que usuários sem acesso root executem comandos que normalmente precisam de privilégios de super usuário. O acesso root ou super usuário é aquele que consegue ter total controle sobre todo o sistema operacional.

O “sudo” sempre virá antes de outro comando, indicando que aquele comando deverá ser executado com privilégios de super usuário.

Ao usar um comando com “sudo” na frente o terminal poderá pedir a senha do usuário atual (a senha usada para iniciar a sessão).

**Usuário root:**

No Ubuntu existe um usuário root que consegue ter acesso e controle total do sistema operacional, por padrão ele vem desativado. Para ativar basta digitar o seguinte comando no terminal:

sudo passwd root

Será pedido para criar e confirmar uma senha para esse usuário.

Para usar o usuário root, basta entrar com ele **no terminal** digitando o seguinte comando:

su root

Digite a senha criada para esse usuário e pronto, a partir daqui qualquer comando dado terá permissão de super usuário.

**Instalação de aplicações**

No Ubuntu existem diferentes formas de instalar aplicações. abaixo serão apresentado algumas delas bem como serão exemplificadas com a instalação de aplicativos que possam ser úteis para trabalhar com robótica no Raspberry pi.

**1.Pacotes Debian:**

Um pacote Debian é uma coleção de arquivos que permite que aplicativos ou bibliotecas sejam distribuídos através do sistema de gerenciamento de pacotes. Esse tipo de pacote é constituído de um único arquivo com a extensão .deb .

Para instalá-lo basta apenas abrir a pasta onde ele está localizado e digitar o seguinte comando:

sudo dpkg -i package\_file.deb

Onde “package\_file.deb” é o nome do arquivo Debian. Note que estamos usando o comando “sudo” então a senha será solicitada para poder conseguirmos finalizar a instalação.

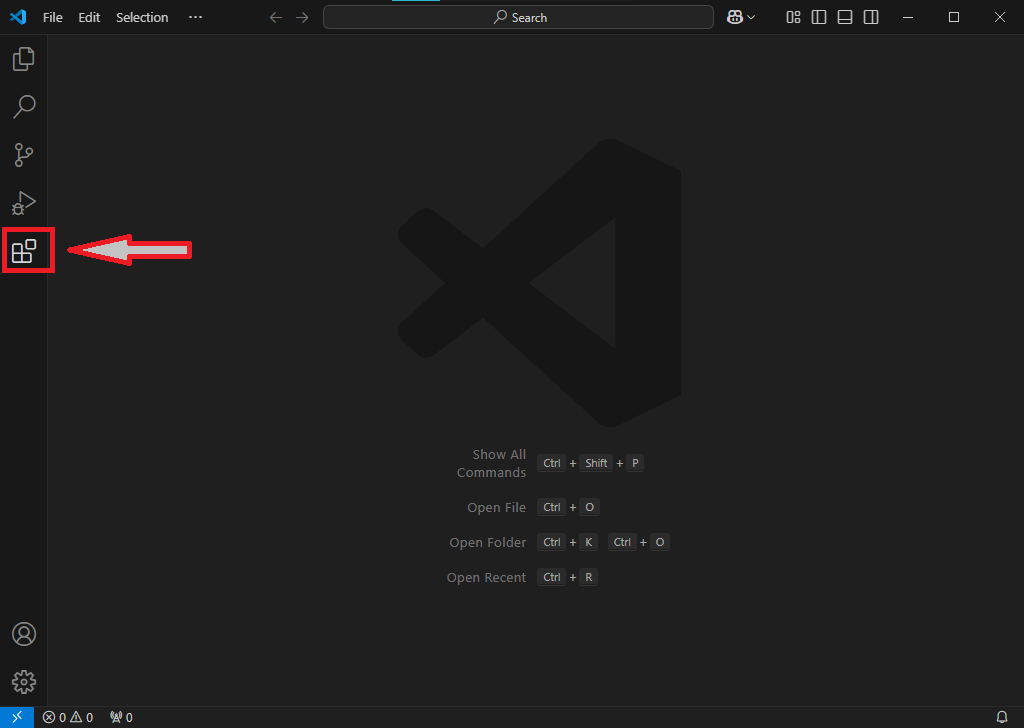
Para desinstalar basta apenas digitar o comando a seguir no terminal, independente do diretório atual:

sudo apt-get remove nome\_do\_pacote

\* ”nome\_do\_pacote” é o nome referente ao Pacote Debian instalado.

\* A senha será solicitada, estamos usando o comando “sudo”.

1.1. Instalando o vscode:

O visual studio code (vscode) é um poderoso ambiente para desenvolvimento de códigos e aplicações. Para instalá-lo no Ubuntu (Raspberry) primeiro deve-se baixar neste link <https://code.visualstudio.com/download> selecionando a opção “.deb” Arm64. Agora siga o passo-a-passo anterior e terá instalado o vscode.

1.2. Configurando o idioma: Por padrão o vscode está em inglês, mas isso pode ser alterado.

1.2.1. Clique em Extensão, se localiza onde a imagem ao lado está indicando:

Um painel lateral irá se abrir, nele pesquise “portuguese” e na seguinte extensão clique em Install:



Após instalado, reinicie o vscode.

(Feche e o abra novamente).

**2. Pacotes tar.xz**

2.1. Descompressão e instalação:

2.1.1. Descompacte o pacote:

Execute o seguinte comando para descompactar o pacote, ele deve ser executado na pasta onde o arquivo “.tar.xz” está localizado:

tar -xJf NomeDoPacote.tar.xz

\*Onde “NomeDoPacote.tar.xz” é o nome do arquivo baixado. Após descompactar uma pasta com o nome do pacote surgirá, abra ela no terminal (pode usar o comando “cd”).

2.1.2. Instalando:

Liste os arquivos da pasta do pacote usando o comando “ls” e caso encontre um arquivo com o nome “install.sh”, execute o seguinte comando:

sudo sh install.sh

\*Usando “sudo”, a senha será solicitada. Caso não exista um arquivo “install.sh” execute os seguintes comandos, na pasta do pacote:

./configure

make

sudo make install

Esses três comandos instalarão o pacote.

2.2. Instalando o arduino IDE:

O arduino IDE é um ambiente de desenvolvimento voltado para a programação de alguns microcontroladores, como Arduino, Esp e STM. Para instalá-lo no Raspberry (Ubuntu), vá ao site oficial de download <https://www.arduino.cc/en/software>, e procure a versão Legacy IDE (1.8.X) no final da página (Apenas esta versão funciona no Raspberry). Na seleção lateral direita selecione a opção Linux ARM 32 bits. O arquivo “.tar.xz” será baixado. Siga o passo-a-passo e você terá instalado o arduino IDE.

Existe um pequeno problema quanto a gravação de um dispositivo através do arduino IDE, ele não tem permissão para acessar as portas USB. Para contornar isso você pode executar o seguinte comando para dar permissão para o arduino IDE (Essa permissão é válida até que o Raspberry seja desligado):

sudo chmod a+rw /dev/ttyUSB0

\*Onde “/dev/ttyUSB0” é o nome da porta USB a qual se quer dar permissão para o arduino IDE.

\*Esse comando usa “sudo”, então a senha será solicitada.

**Instalando o FTP**

Um servidor FTP é uma aplicação que permite que os arquivos do raspberry possam ser acessados, alterados, copiados e até apagados por outro dispositivo com maior facilidade de que o ssh. Aqui iremos usar o vsFTPd.

Para instala-lo use o comando:

sudo apt-get install vsftpd

Você receberá uma solicitação de confirmação, basta digitar y e enter. Após instalar reinicie o Raspberry. Agora verifique se o serviço referente a ele já está rodando, para isso use:

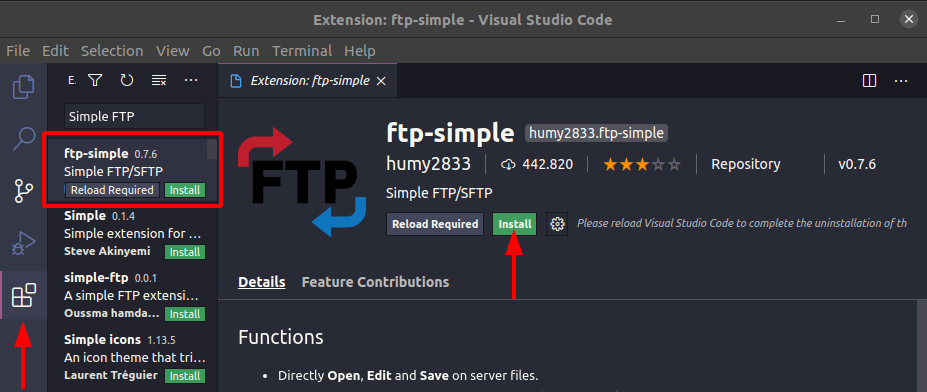
sudo systemctl status vsftpd

Agora abra o arquivo de configuração localizado em  */etc/vsftpd.conf*  e descomente a linha ‘*write\_enable=YES*’, ela permite que o usuaria adicione arquivos. Salve e feche. Este arquivo deve ser aberto com permissão de superusuario. Após isso reinicie o serviço do vsftpd com o comando abaixo:

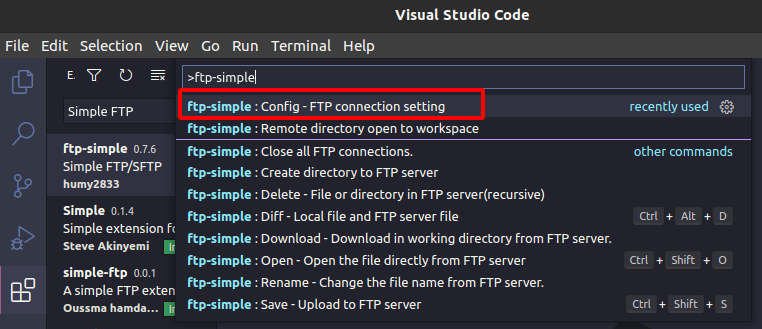
sudo systemctl restart vsftpd

Acessando através do VSCode:

Instale a extensão ‘ftp-simple’.



Depois de intalar, pressione F1 e digite ‘ftp-simple’, em seguida clique na opção mostrada abaixo:



Isso abrirá um arquivo com as informações de acesso do ftp.

* Em **name**, dê um nome a sua conexão, o nome é de sua escolha.
* Em **host** digite o endereço ip do dispositivo.
* Em **username** e **password** digite o nome do usuario e a senha respectivamente.

Após isso reinicie o vscode. Aperte F1, digite ftp-simple e selecione a opção “*Remote directory open to workspace*”, selecione a pasta que deseja abrir e pronto, você consegue editar, criar e excluir arquivos.

**Serviços**

Um serviço é um programa que roda em segundo plano fora do controle interativo dos usuários do sistema, por falta de uma interface. Eles são executados durante a inicialização do sistema, em diferentes etapas. No linux existem vários serviços que já estão lá para auxiliar o próprio sistema operacional, mas você pode também criar um serviço personalizado que faça algo que você precise.

**Criação:**

1.Crie o arquivo:

Navegue até a pasta  */etc/systemd/system*  e crie um arquivo com a extensão “.service”. Por exemplo ***myservice.service***, este é um arquivo Unit, arquivo de configuração de um serviço.

2.Edite o Arquivo: Abra ele em um editor de texto. E coloque o seguinte *texto mínimo*:

| [Unit]  Description=Meu primeiro servico  After=network.target  [Service]  Restart=always  RestartSec=1  User=usuario  ExecStartPre=/usr/bin/sh ~/caminho/script.sh  ExecStart=/usr/bin/python3 /caminho/arquivo.py    [Install]  WantedBy=multi-user.target |
| --- |

Description A descrição do seu serviço. Pode ter qualquer valor

After O ponto da inicialização onde o serviço vai ser iniciado. Alguns valores são:

***network-pre.target*** Antes de qualquer interface de rede ser configurada

***network.target*** Quando as configurações de rede estão feitas.

***network-online.target***  Quando a rede estiver ativa, conectada.

***default.target***  Quando o sistema estiver iniciado.

Além disso, pode ser indicado um outro serviço para que o serviço atual seja executado após o serviço referenciado.

Restart Em caso de erro, indica se deve reiniciar o serviço (***always*** sempre).

RestartSec Quanto tempo, em segundos, deve se esperar para reiniciar em caso de erro.

User O usuário que está executando o serviço. Pode ser o nome do seu usuário padrão ou o usuário ***root***.

ExecStartPre script a ser executado imediatamente antes da execução do serviço.

ExecStart script a ser executado pelo serviço.

WantedBy Importante para que o seu serviço consiga iniciar automaticamente durante a inicialização sem um requisição por parte de outro serviço. Por padrão usamos o valor ***mult-user.target*** (Esse é o ponto do sistema que se assemelha ao *default.target)*.

Pronto! O seu serviço já está criado. Para iniciá-lo e habilita-lo, veja a lista de comandos abaixo.

**Comandos:**

Existem alguns comandos responsáveis por gerenciar os serviços, eles pertencem ao ***systemd***, o sistema de inicialização do linux. Por manipular recursos fundamentais para a execução do sistema, todos estes comandos precisam de permissões de super usuário, por tanto devem ser seguidos por um ***sudo*** ou executados por um usuário root. Os comandos abaixo devem estar no seguinte padrão:

*sudo* ***systemctl*** *<comando> <?parâmetro>*

\*Usa ***sudo,*** pedirá senha. Aqui estão os comandos:

*daemon-reload* Recarrega as configurações do *systemd*, deve ser executado toda vez que um arquivo de serviço for modificado. Não precisa de parâmetro.

*start*  Inicia o serviço especificado. O parâmetro deve ser o nome do arquivo “*.service”* relacionado com o serviço.

*stop* Encerra o serviço especificado. O parâmetro deve ser o nome do arquivo “*.service”* relacionado com o serviço.

*restart* Reinicia o serviço especificado. O parâmetro deve ser o nome do arquivo “*.service”* relacionado com o serviço.

*status* Mostra informações sobre o serviço especificado. Diz se ocorreu um erro, por exemplo. O parâmetro deve ser o nome do arquivo “*.service”* relacionado com o serviço.

*enable* Habilita o serviço para que este possa ser iniciado sempre, automaticamente, durante as inicializações do sistema. O parâmetro deve ser o nome do arquivo “*.service”* relacionado com o serviço.

*disable* Desabilita o serviço para que este não seja iniciado durante as inicializações do sistema. O parâmetro deve ser o nome do arquivo “*.service”* relacionado com o serviço.

*mask* Mesmo desabilitado, alguns serviços ainda podem ser iniciados durante a inicialização do sistema se foram requeridos por outros serviços. Para evitar isso, caso necessário, este comando mascara o acesso a inicialização automática do serviço, só sendo possível iniciá-lo manualmente. O parâmetro deve ser o nome do arquivo “*.service”* relacionado com o serviço.

*unmask* desmascara o serviço especificado. O parâmetro deve ser o nome do arquivo “*.service”* relacionado com o serviço.

**Executando scripts:**

1. Scripts bash:

O Bash é um interpretador de linha de comando para sistemas Linux. Ele atua como uma interface entre o usuário e o sistema operacional, permitindo a execução de comandos e scripts para realizar várias atividades. A maioria dos comandos executados diretamente no terminal são esse tipo de comando.

Para execultar um comando desse tipo em um serviço basta usar a seguinte sintaxe:

***ExecStart*** = /bin/bash -c ‘***comando\_bash***’

\*Onde ***comando\_bash*** é o comando que deseja executar.

Uma observação importante é que caso queira pode usar essa sintaxe na propriedade *ExecStartPre* também.

Para usar um comando com permissão de super usuário, o serviço deve pertencer ao usuário ***root***.

2. Script python:

Para executar um script python deve-se usar a seguinte sintaxe:

***ExecStart*** *= /bin/python3* ***/caminho/do/codigo.py***

\*Onde ***/caminho/do/código.py*** é o caminho completo do código python

\*Este exemplo refere-se a versão 3.12 do python

Vale ressaltar que ***/bin/bash*** e ***/bin/python3*** são o local dos intérpretes dos respectivos tipos de scripts.

**Python**

Por padrão o python já vem instalado no Ubuntu, e no caso do Ubuntu 24.04 LTS temos já instalada a versão 3.12. Aqui serão apresentadas algumas bibliotecas que podem ser úteis no python, principalmente trabalhando-se com o ROS e com a robótica.

**threading**:

Essa biblioteca permite a realização de tarefas simultâneas. Vem instalada no python por padrão.

Exemplo de uso:

import time

from threading import Thread

def carro1():

for i in range(5):

print("Carro 1 está correndo...")

time.sleep(1)

def carro2():

for i in range(5):

print("Carro 2 está correndo...")

time.sleep(1)

# Criando threads

t1 = Thread(target=carro1)

t2 = Thread(target=carro2)

# Iniciando threads

t1.start()

t2.start()

# Esperando as threads terminarem

t1.join()

t2.join()

A função *carro1* e *carro2* serão executadas simultaneamente.

**rich:**

Essa é uma biblioteca que permite escrever texto estilizado e tabelas no console do terminal. Vem instalada no python por padrão.

Exemplo de exibição de duas tabelas lado a lado sendo atualizada 3 vezes por segundo:

from rich.console import Console

from rich.table import Table

from rich.layout import Layout

import time

console = Console()

def update\_table(data, nome):

table = Table(title=nome) # Cria uma tabela e dá um titulo a ela

table.add\_column("REF", justify="center", style="cyan") # Adiciona uma coluna

table.add\_column("DATA", justify="center", style="magenta") # Adiciona coluna

for row in data:

table.add\_row(\*row) # Adiciona uma linha

return table

data\_sensores = [

["S0", "0cm"],

["S1", "12cm"],

["S2", "15cm"]

] # data tabela 1

data\_motores = [

["MT1", "200rpm"],

["MT2", "127rpm"]

] # data tabela 2

i = 0

while True:

console.clear() # limpa o console

layout = Layout() # cria um layout

layout.split\_column(Layout(name="top")) # adiciona uma coluna ao layout

layout["top"].split\_row(

update\_table(data\_sensores, "sensores"),

update\_table(data\_motores, "motores")

) # adicion aas duas tabelas em uma unica linha

console.print(layout) # desenha as tabelas

time.sleep(0.3) # taxa de atualização (3 vezes por segundo)

data\_sensores[0][1] = str(i)+"cm" # atualiza os valores

i+=1

Para mais detalhes consulte a documentação. Link na seção **REFERÊNCIAS**.

**pySerial:**

Esta biblioteca permite a comunicação serial através de uma porta USB ou através da UART do Raspberry. Ela já vem instalada no python.

Exemplo de uso enviando dados e lendo quando disponíveis.

import serial

ser = serial.Serial('/dev/ttyAMA0', 9600, timeout=1)

# porta, taxa de transmissão, tempo maximo de espera para uma resposta (em segundos)

ser.write(b"Hello world") #escreve na serial

# string codificada ou bytearray

while True:

if ser.in\_waiting>0: # ver a quantidade de dados disponiveis para leitura

lido = ser.read(ser.in\_waiting) # ler a quantidade de dados pedido

print(lido)

Para mais detalhes consulte a documentação. Link disponível na seção **REFERÊNCIAS**.

**subprocess:**

Essa biblioteca permite que o python execute comando de terminal. Ela já vem instalada no python.

Exemplo de uso:

import subprocess

resposta = subprocess.run(

"ls ./", # comando

shell=True, # indica comando em uma unica string

text=True, # garante que a resposta seja um texto

capture\_output=True # captura a saída

)

print(resposta.stdout) # exibe a saida no terminal

Para mais detalhes consulte a documentação. Link disponível na seção **REFERÊNCIAS.**

**socket:**

Esta biblioteca permite que diferentes scripts python que estejam rodando na mesma máquina se comuniquem entre si. Ela já vem instalada por padrão.

Exemplo no qual um script envia um dado para o outro:

Arquivo 1:

# Fala

import socket

import time

sock = socket.socket( socket.AF\_INET, socket.SOCK\_DGRAM)

i = 0

while True:

sock.sendto(str(i).encode(), ( "127.0.0.1", 5000))

# string codificada a ser enviada,

# tupla com o ip padrão de comunicação interna e a porta na qual ela vai ocorrer

print("Enviado: ", i)

time.sleep(1)

i += 1

Arquivo 2:

# Escuta

import socket

sock = socket.socket( socket.AF\_INET, socket.SOCK\_DGRAM)

sock.bind(

( "127.0.0.1", 5000)

# IP padrão de comunicação interna, porta para comunicação

) # função para dizer de onde ele deve ouvir

while True:

data, addr = sock.recvfrom(1024) # espera um dado ser recebido

rcv = data.decode() #decodifica a string recebida

print("Recebido: ", rcv)



GitHub

jh

****

**GPIOs**

O Raspberry pi 4 model B possui um barramento de pinos com diversas funções, entre comunicação I2C, UART e Alimentação existem as GPIOs e Entradas/Saídas digitais de uso geral. Uma GPIO pode ser configurada como entrada ou saída e um nível pode ser atribuído ou lido. Vale ainda ressaltar que esses pinos, assim como todos os outros, trabalham apenas com 3V de tensão.

**Instalando as GPIOs:**

Para usar as GPIOs do Raspberry é necessário instalar uma biblioteca chamada lgpio. Para instalar ela basta executar o seguinte comando:

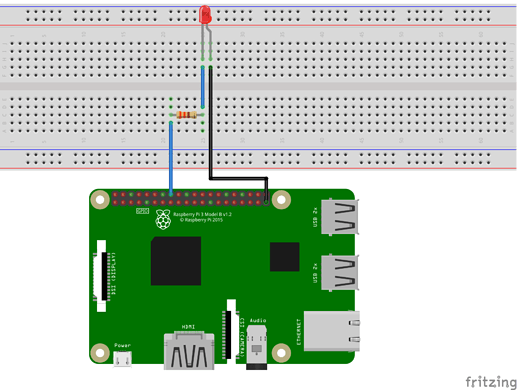
*sudo apt install python3-lgpio*

Após ter instalado as GPIOs já podem ser manipuladas no python.

**Saídas digitais:**

Estas são usadas para fornecer um nível alto (3V) ou um nível baixo (0V). Ela pode acender leds ou até mesmo controlar motores. Abaixo está um código python e o esquema de ligação que faz um led piscar no GPIO 23:

Conexão:



Código:

Este código faz o led piscar

# Piscando um LED no pino 23

import time

import lgpio

LED = 23 #pino do LED

h = lgpio.gpiochip\_open(0) # Abre o chip dos GPIOS, cria uma referência dele

lgpio.gpio\_claim\_output(h, LED) # Define o pino como saída

try:

while True:

lgpio.gpio\_write(h, LED, 1) # Acende o Led

time.sleep(1)

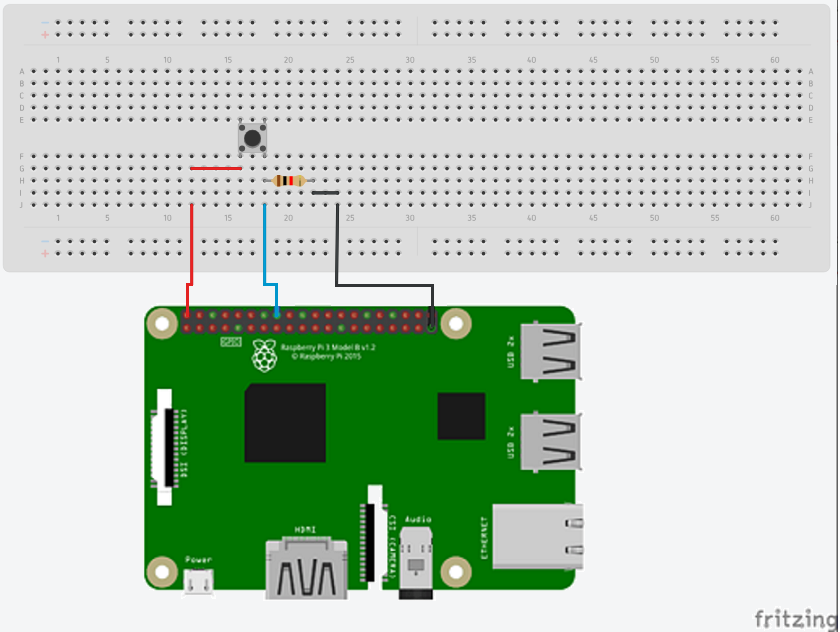
lgpio.gpio\_write(h, LED, 0) # Apaga o Led

time.sleep(1)

except KeyboardInterrupt:

lgpio.gpio\_write(h, LED, 0) #apaga o led

lgpio.gpiochip\_close(h)



**Entradas digitais:**

São usadas para botões ou sensores. O dispositivo conectado a ela pode fornecer um nível de tensão, 3V ou 0V, que é lido pelo Raspberry, como 0 ou 1, e reage de acordo com o valor. A seguir um código e ao lado a ligação esquemática do uso de uma botão em uma entrada digital, o resistor é de 1k, mas pode-se usar um de 10k:

Código:

# Botão no pino 23

import lgpio

BOTAO = 23 #pino do BOTÃO

h = lgpio.gpiochip\_open(0) # Abre o chip dos GPIOS, cria uma referência dele

lgpio.gpio\_claim\_input(h, BOTAO) # Define o pino como entrada

try:

while True:

if(lgpio.gpio\_read(h, BOTAO)==1):

print("Você pressionou o botão")

except KeyboardInterrupt:

lgpio.gpiochip\_close(h)#fecha o chip do gpio



**UART**

**O que é UART:**

No barramento de pino do Raspberry Pi 4 existem pinos marcados como RX e TX, eles fazem parte da UART. Ela é um tipo de comunicação em que dois dispositivos trocam dados entre si. Ela não necessita de um sinal de clock para sincronizar a transmissão, só é necessário que ambos os dispositivos estejam se comunicando na mesma velocidade.

São necessárias duas conexões para criar a comunicação, temos os pinos Rx e Tx e vamos imaginar uma comunicação entre um dispositivo A e B. A ligação entre eles deve ser a seguinte, Rx do dispositivo A ligado ao Tx do dispositivo B, Rx do dispositivo B ligado ao Tx do dispositivo A. O pino Rx é o receptor e por tanto uma entrada digital, enquanto Tx é um transmissor e, logo, uma saída digital.

**Configurando a UART:**

No Raspberry Pi 4 a UART não vem habilitada para uso, e por isso devemos primeiramente habilita-la. Por padrão ela é usada para o Raspberry se comunicar internamente com seu módulo bluetooth, por isso também devemos desativá-lo.

1. Habilitando a UART no arquivo de inicialização:

Abra o arquivo ***“config.txt”***localizado em *“/boot/firmware”*. Deve ser aberto com permissão de superusuário. Recomendo abrir no editor **nano** através do comando:

*sudo nano /boot/firmware/config.txt*

Nas primeiras linhas, logo após ***[****all* ***]***, adicione “**enable\_uart=1**”, sem as aspas.

2. Desabilitando o bluetooth:

Também no arquivo ***“config.txt”***localizado em *“/boot/firmware”* adicione a linha “**dtoverlay=disable-bt**”, sem aspas. Essa linha pode ficar abaixo da linha adicionada anteriormente. Agora devemos desativar o serviço responsável pelo bluetooth. Ele se chama “**bluetooth.service**”. Para desabilitar primeiro vamos pará-lo, com o comando abaixo:

*sudo systemctl stop* ***bluetooth.service***

Em seguida, desabilitamos o serviço:

*sudo systemctl disable* ***bluetooth.service***

Para garantir que o serviço não será iniciado por outro, vamos mascarar ele:

*sudo systemctl mask* ***bluetooth.service***

3. Desativando o console serial:

Ao desativar o bluetooth, a UART é redirecionada para o uso do console serial, para desativar isso basta desabilitar o console serial. O serviço responsável pelo console serial é o “**serial-getty@ttyAMA0.service**”.

Primeiro pare este serviço:

*sudo systemctl stop* ***serial-getty@ttyAMA0.service***

Em seguida, desabilite o serviço:

*sudo systemctl disable* ***serial-getty@ttyAMA0.service***

E depois mascare-o:

*sudo systemctl mask* ***serial-getty@ttyAMA0.service***

Reinicie o dispositivo e a UART estará disponível em “**/dev/ttyAMA0**”. No entanto, ela só poderá ser lida ou escrita pelo super usuário.

**Tornando-a acessível:**

Para fazer com que o usuário comum consiga acessar e manipular a UART basta mudar as configurações por meio do comando:

*sudo chmod a+rw /dev/ttyAMA0*

Agora a UART pode ser acessada por qualquer usuário. No entanto após a reinicialização as configurações da UART são redefinidas e por isso é necessário que esse comando seja inicializado toda vez que o sistema for iniciado. Para isso podemos criar um serviço\*\* que execute o comando bash “*chmod a+rw /dev/ttyAMA0*” e que pertença ao usuário root. Antes disso temos que habilitar o usuário root\*.

\*Para habilitar o usuário root veja o Capítulo 1 na parte **Comandos do ubuntu.**

\*\*Para criar um serviço veja no Capítulo 1 na parte **Serviços**.

**Usando a UART:**

Abaixo está um exemplo de como enviar dados e ler a UART usando a biblioteca *pyserial* do python.

import pyserial

import time

ser = serial.Serial("/dev/ttyAMA0",9600, timeout=1)

# "/dev/ttyAMA0" é o local referente a UART

# o segundo parametro é a taxa de transmissão bits por segundos

# timeout é o tempo máximo de espera por uma leitura (em segundos)

ser.write(b'Hello World') # Escreve Hello Word na serial

time.sleep(1) # Espera 1s

if ser.in\_waiting>0: # Se existe bytes de entrada a serem lidos

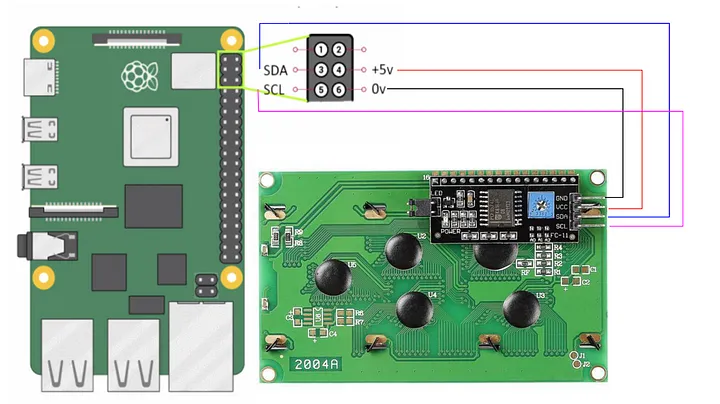
print(ser.readline()) # Ler a UART e printa o valor



**LCD**

**Conexão física:**

Aqui apresentaremos o display lcd 16x2 com módulo I2c. Abaixo segue a conexão física:



**Bibliotecas:**

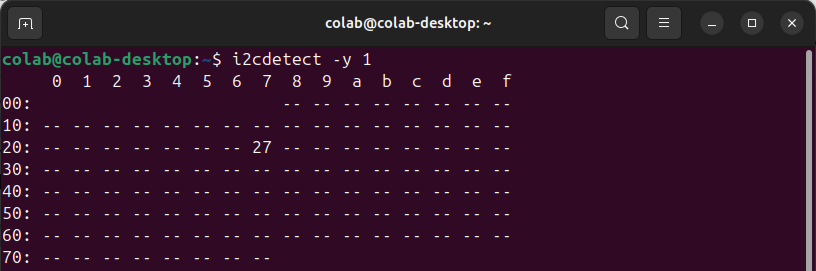
Após conectá-lo deve-se instalar a ferramenta de detecção de dispositivos i2c, através do comando:

*sudo apt install i2c-tools*

Em seguida verifique o endereço do seu display usando o seguinte comando:

*i2cdetect -y 1*

O resultado deve ser o seguinte:



O endereço que apareceu, “27” neste caso, é o endereço do seu display.

Agora instale a biblioteca smbus através do comando abaixo:

*sudo apt install python3-smbus*

**Driver de controle:**

Para manipular o LCD é necessário enviar uma série de comandos pré-definidos. Para evitar que todas vez que você for fazer algo tenha de escrever em seu código os bits de controle e comando, crie um arquivo chamado I2C\_LCD\_driver.py, que servirá como base de controle, e coloque oconteudo do arquivo *RPi\_I2C\_driver.py* disponível em <https://gist.github.com/DenisFromHR/cc863375a6e19dce359d>. Esse é o conjunto de instruções necessárias para o funcionamento adequado do LCD.

**Funções básicas:**

Vamos falar das funções básicas presente no arquivo:

*background* Esta função liga ou desliga a luz de fundo do display. Seu parâmetro deve ser 0 para luz apagada e 1 para luz acesa.

*lcd\_display\_string* Esta função exibe uma string em uma data posição. Seus parâmetros são, nesta ordem, a string, a linha (1 ou 2) e a coluna (começando em 0).

Abaixo está um exemplo de aplicação desse driver:

import I2C\_LCD\_driver

import time

lcdi2c = I2C\_LCD\_driver.lcd()

lcdi2c.lcd\_display\_string("EXEMPLO", 1,0) #linha 1 coluna 1

lcdi2c.lcd\_display\_string("LCD", 2, 5) #linha 2 coluna 6

while True:

lcdi2c.backlight(0)#apaga luz de fundo

time.sleep(1)

lcdi2c.backlight(1)#acende luz de fundo

time.sleep(2)

Este exemplo exibe alguns textos e faz a luz de fundo piscar.

**Câmera**



**Introdução**



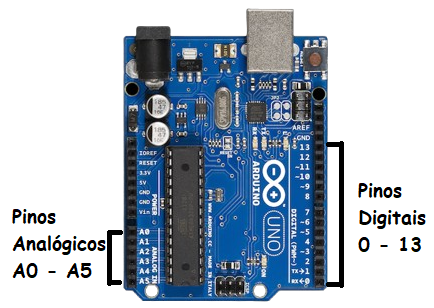
**Arduino**

**Introdução:**

O arduino é uma plataforma de prototipagem baseada no microcontrolador Atmega. Ela possui várias portas que podem ser pinos digitais, pinos analógicos, pinos de comunicações específicas e/ou pinos de alimentação. Dependendo do modelo podem haver quantidades de pinos diferentes, memória diferente, processamento diferente, tamanhos diferentes, etc. Seu processo de programação é simples, sendo necessário apenas instalar o *ARDUINO IDE* no seu dispositivo, conectar a placa ao computador através de um cabo USB e pronto, está pronto para ser programado. O código em si também é algo muito simples, usando a linguagem C++, uma linguagem de alto nível de fácil compreensão. Aqui será tratado apenas sobre o Arduino UNO, a diferença entre outras versões, em questão de programação, está apenas na quantidade de pinos.

**GPIOs:**

GPIOs são pinos digitais de uso geral. Eles podem ser configurados como saídas, podendo fornecer tensões de 5V ou 0V, ou entradas, podendo reconhecer tensões próximas de 0V ou 5V. Abaixo estão os pinos do Arduino UNO:



Veja que ele possui 14 pinos digitais (Esses pinos são exclusivamente digitais) e 6 pinos analógicos (serão apresentados mais à frente), que também podem ser digitais.

Programação dos pinos:

É importante sabermos que por padrão um novo programa vem com duas funções já pré-definidas, o *void setup* e o *void loop*. O *void setup* é executado apenas uma vez durante a inicialização, enquanto o *void loop* fica sendo executado infinitamente em loop. Para programar uma porta devemos primeiro definir se ela será uma entrada ou saída. Fazemos isso usando o comando *pinMode* onde devemos especificar o número do pino que queremos definir e se ele vai ser **OUTPUT** (saída) ou **INPUT** (entrada). Abaixo está um exemplo de como definir o pino 13 como saída:

pinMode(13, OUTPUT);



Abaixo um outro exemplo de como definir o pino 9 como entrada:

pinMode(9, INPUT);



Para definir um dos pinos analógicos como pinos digitais basta usar no lugar do número, seu nome (A0, A1, A2, A3, A4, A5).

Este comando deve ficar dentro do *void setup*.

Alterando saídas:

Com um pino definido como **OUTPUT** podemos alterar seu valor de tensão entre 5V e 0V. Para isso devemos usar o comando *digitalWrite* e passar o número do pino e o valor 0, para 0V, ou 1, para 5V. Exemplo alterando o pino 13 para 5V:

digitalWrite(13, 1);



Recomendo o uso desse comando no *void loop*.

Piscando um led:

Abaixo tem um circuito que faz o led do pino 13 piscar. Esse led já está integrado no arduino.

void setup(){

pinMode(13, OUTPUT);

}

void loop(){

digitalWrite(13, 1);

delay(1000); //espera 1 segundo (1000 miléssimos)

digitalWrite(13, 0);

delay(1000); //espera 1 segundo (1000 miléssimos)

}



O comando *delay* faz com que o código pare de ser executado durante o tempo que foi passado a ele (em milésimos de segundos).

Lendo entradas:

Com um pino definido como **INPUT**, pode se ler o nível de tensão aplicado nele através do comando *digitalRead* no qual deve ser especificado o pino a ser lido. O valor retornado será 0 (para 0V) ou 1 (para 5V). Exemplo, lendo o pino 10 e salvando em uma variável:

int leitura = digitaRead(10);

Abaixo um exemplo da leitura de entradas. Será lido um valor de um botão e, se ele for apertado, acenderá o led do pino 13.

O circuito:



O código:

void setup()

{

pinMode(5, INPUT);

pinMode(13, OUTPUT);

}

int led = 0;

void loop()

{

if(digitalRead(5)==1){

led = !led;

delay(500);

}

digitalWrite(13, led);

}



**PWM:**

PWM (Modulação por Largura de Pulso - **P**ulse **W**idth **M**odulation) é uma técnica usada para alterar a tensão média sobre uma carga e alterar assim sua potência. Em motores DC seu efeito é o controle sobre sua velocidade de rotação, em leds controla-se seu brilho.

Esse processo consiste em aplicar um trem de pulso (sequência de pulsos quadrados) com uma frequência constante e alterar o tempo em que o pulso se mantém em nível alto, alterando assim a potência sobre a carga.

O arduino possui por padrão suporte ao pwm que pode ser facilmente usado. No caso do arduino Uno apenas 6 pinos suportam pwm, sendo eles os pinos 3, 5, 6, 9, 10, 11. Para usar no código o PWM, basta definir o pino como saída e para dar a ele um nível de potência usar o comando abaixo:

analogWrite(5, valor);

Neste exemplo usa-se o pino 5. “valor” deve ser um número entre 0 e 255 onde 0 representa 0% da potência máxima e 255 representa 100% da potência máxima.

Abaixo um exemplo que acende e apaga um led gradativamente usando PWM. O led deve estar conectado no pino 5.

void setup()

{

pinMode(5, OUTPUT);

}

void loop()

{

//aumenta o brilho

for(int i = 0; i<=255; i++){

analogWrite(5, i);

delay(10);

}

//abaixa o brilho

for(int i = 255; i>=0; i--){

analogWrite(5, i);

delay(10);

}

delay(300);

}



**Timers:**

Um timer é um contador que existe dentro do microcontrolador que pode permitir a realização de uma dada tarefa após atingir um dado valor ou após atingir seu valor máximo.

O Arduino Uno possui 3 timers, 2 de 8 bits (TIMER0 e TIMER2) e 1 de 16 bits (TIMER1). No entanto, alguns desses timers são usados internamente para algumas funções específicas, como gerar sinais pwm, função *delay*, entre outros. Alterar o modo de operação deles pode influenciar no funcionamento do seu código. O TIMER1 é o timer que é usado em algumas bibliotecas muito específicas e usá-lo dificilmente vai impactar em outras funções.

Para usar e configurar o TIMER1, precisaremos configurar alguns registradores, sendo eles:

TCCR1A Esse registrador juntamente com o TCCR1B configura o TIMER1, este configura o modo de operação, definimos seu valor em 0 para modo de operação normal.

TCCR1B Configura o modo de clock do timer e outras configurações e junto com o TCCR1A configura o tipo de interrupção do TIMER1.

TCNT Guarda a contagem do timer.

TIMSK1 Habilita a interrupção do TIMER1, o bit TOIE1 habilita a interrupção por estouro, o bit OCIE1A habilita a interrupção por comparação com o registrador OCR1A.

OCR1A Salva o valor alvo para estouro do TIMER1 por comparação.

Para mais informações consulte o datasheet do Atmega328p.

Funções de disparo:

Para a interrupção por estouro use:

ISR(TIMER1\_OVF\_vect){}



Para a interrupção por comparação com o OCR1A use:

ISR(TIMER1\_COMPA\_vect){}

Abaixo segue um exemplo que faz um led piscar a cada um segundo com o uso do TIMER1:

#define ledPin 13

void setup()

{

pinMode(ledPin, OUTPUT);

// Configuração do TIMER1

TCCR1A = 0; //confira timer para operação normal

TCCR1B = 0; //limpa registrador

// modo CTC (WGM12 = 1), prescaler de 1024: (CS12 = 1 e CS10 = 1)

//prescaler = clock do arduino(16Mhz)/1024

TCCR1B |= (1 << WGM12)|(1<<CS10)|(1 << CS12);

TCNT1 = 0; //zera temporizado

OCR1A = 0x3D09; // carrega registrador de comparação:

// (16MHz/1024)/1Hz = 15625 = 0X3D09 (1Hz = 1 vez por segundo)

TIMSK1 |= (1 << OCIE1A);

// habilita interrupção por comparação com OCR1A

}

void loop()

{

//loop principal. a manipulação do led é feita na ISR

}

byte led = 0;

ISR(TIMER1\_COMPA\_vect)// interrupção comparação de TIMER1 com OCR1A

{

digitalWrite(ledPin, led);

led = !led; // inverte o led

TCNT1 = 0; // zera o timer

}



**Interrupções Externas:**

São funções disparadas com base em uma condição de estado de um pino digital. Elas são disparadas imediatamente quando a condição pré-definida é alcançada, interrompendo o processo principal (*void loop*). Sua sintaxe é:

attachInterrupt(digitalPinToInterrupt(pino), ISR, modo);



Onde:

* *pino* é o número do pino no qual a interrupção externa está entrelaçada. No arduino uno apenas os pinos 2 e 3 podem ser usados como interrupção externa.
* *ISR* é o nome da função que será disparada ao se atingir a condição especificada.
* *modo* é o tipo de condição que dispara a interrupção. Eles podem ser:
  + **LOW** : Aciona a interrupção quando o estado do pino for 0.
  + **CHANGE** : Aciona a interrupção quando o estado do pino mudar.
  + **RISING** : Aciona a interrupção quando o estado do pino for de LOW para HIGH apenas.
  + **FALLING** acionar a interrupção quando o estado do pino for de HIGH para LOW apenas.

Abaixo um exemplo de interrupção durante uma mudança de 0 para 1 para acender e apagar o led do pino 13. O circuito consiste em um botão no pino 3 com um resistor de pull down:

void setup(){

pinMode(13, OUTPUT);

pinMode(3, INPUT);

attachInterrupt(digitalPinToInterrupt(3), botao, RISING);

}

int led = 0;

void loop(){

// nada acontece

}

//ISR

void botao(){

digitalWrite(13, led);

led = !led; //inverte o led

}



**Pin Change Interrupt:**

É bem semelhante a interrupção externa, no entanto ela é uma função que é SEMPRE disparada quando um pino muda de estado. Além disso, ela é uma função coletiva, ou seja, ela monitora um conjunto de pinos e dispara a mesma função para a mudança de qualquer um deles. Diferente das interrupções externas, todos os pinos do arduino uno podem ser usados aqui, sendo divididos em 3 conjuntos.

Para usá-lo basta configurar alguns registradores, sendo eles:

* **PCICR** - Ele habilita ou desabilita o pin change interrupt de um grupo de pinos, para isso os pinos são divididos em 3 grupos, sendo eles:
  + PORTB (pinos 8 a 13) : o Pin Change ele é ativado pelo primeiro bit do registrador (bit denominado PCIE0).
  + PORTC (pinos A0 a A5) : o Pin Change ele é ativado pelo segundo bit do registrador (bit denominado PCIE1)
  + PORTD (pinos 0 a 7) : o Pin Change ele é ativado pelo terceiro bit do registrador (bit denominado PCIE2)
* **PCMSK0, PCMSK1, PCMSK2** - Eles controlam quais pinos dentro do PORTB, PORTC e PORTD respectivamente, podem gerar uma interrupção por mudança de estado (Pin Change Interrupt).
  + PCMSK0 - controla PORTB, sendo o bit 0 define o pino 8, e o bit 5 o pino 13.
  + PCMSK1 - controla PORTC, sendo o bit 0 define o pino A0, e o bit 5 o pino A5.
  + PCMSK2 - controla PORTD, sendo o bit 0 define o pino 0, e o bit 7 o pino 7.

Ao sofrer uma mudança de estado, qualquer pino de um mesmo PORT dispara a mesma ISR, sendo elas, para cada PORT:

* ISR(PCINT0\_vect) - Para qualquer pino do PORTB.
* ISR(PCINT1\_vect) - Para qualquer pino do PORTC.
* ISR(PCINT2\_vect) - Para qualquer pino do PORTD.

Abaixo segue um exemplo, habilitando o Pin Change Interrupt apenas nos pinos 10 e 11, onde deve haver um botão com resistor de pull down deve estar em cada pino:

void setup(){

pinMode(13, OUTPUT); // led da placa

PCICR |= (1 << PCIE0); // interrupção para os pinos D8 a D13 (PORTC)

// habilita interrupção nos pinos 10 e 11

PCMSK0 |= (1 << PCINT3) | (1 << PCINT2); // 2-10, 3-11

}

void loop(){

// nada acontece

}

//ISR

ISR(PCINT0\_vect){

if(digitalRead(10)){

// interrupção do pino 10 (botão apertado)

digitalWrite(13, 1); // liga o led

}

if(digitalRead(11)){

// interrupção do pino 11 (botão apertado)

digitalWrite(13, 0); // desliga o led

}

}



STM32

**Introdução:**

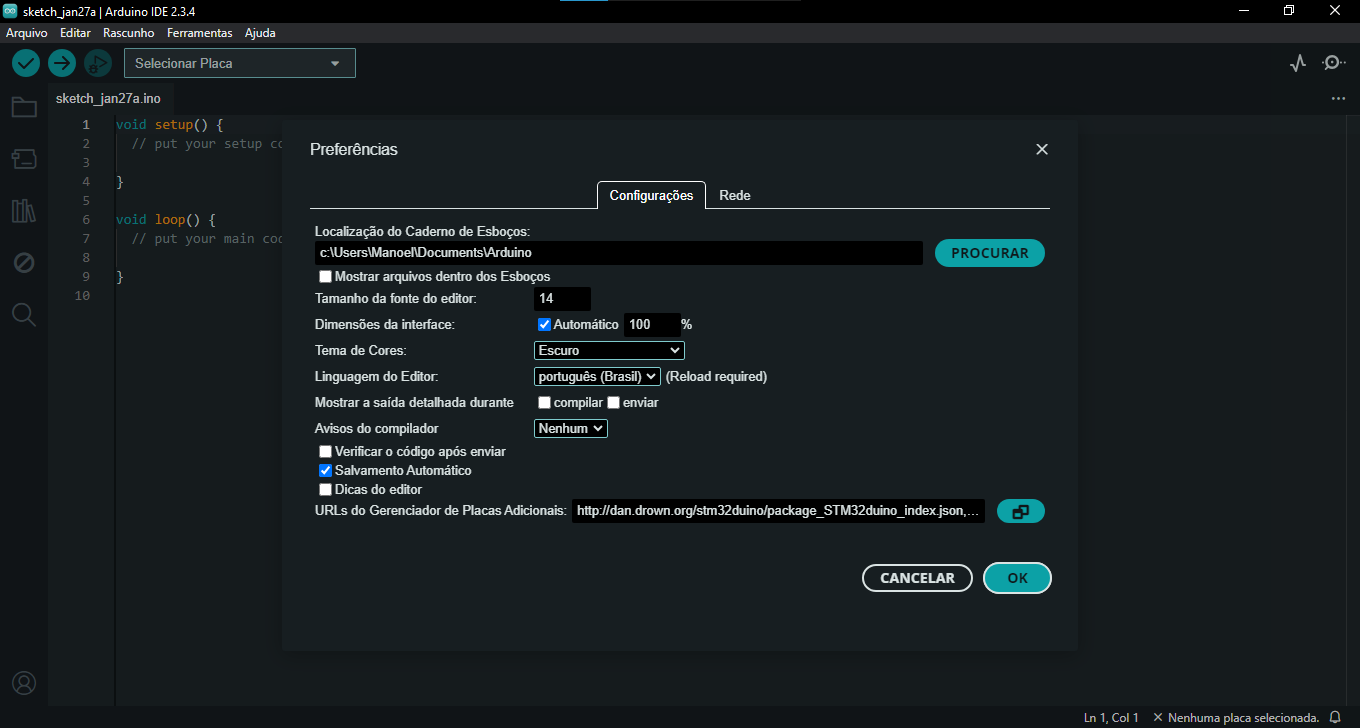
O STM32 é uma família de microcontroladores de 32 bits que passou a ser usada em placas de prototipagem. Aqui abordaremos a placa blue pill que possui o STM32F103C8T6, tem vários recursos e um pequeno tamanho.

**Programação:**



Uma coisa importante sobre a blue pill é que, mesmo tendo uma entrada usb, ela não pode ser programada conectando diretamente a um computador, para isso é necessário um programador externo. Aqui apresentaremos o uso do ST-LINK.

Sua programação pode ser feita através do *ARDUINO IDE*, basta primeiramente baixar o programa que se comunica com o ST-LINK e depois configurar a IDE. Para isto, acesse o link: <https://www.st.com/en/development-tools/stsw-link004.html> do site oficial da ST e faça o download do ST-Link Utility. Após baixar, instale em seu computador. Agora vamos para o ARDUINO IDE:



No menu superior clique em **ARQUIVO > PREFERÊNCIAS** e a seguinte tela vai aparecer:

No campo “URLs do Gerenciador de Placas Adicionais” adicione a seguinte URL: http://dan.drown.org/stm32duino/package\_STM32duino\_index.json

Após isso, vá em **FERRAMENTAS > PLACA > GERENCIADOR DE PLACAS…** na janela que abrir pesquise por STM32 e instale a seguinte placa:

Agora para conseguir programar seu stm através do ARDUINO IDE, vá em **FERRAMENTAS** e modifique os seguintes parâmetros:

* PLACA “*Generic STM32F103C Series*” ou “*Generic STM32F103C6/fake STM32F103C8*” se sua placa não for a original.
* Upload method ST-Link

Com isso você já pode conectar sua placa com o ST-Link e em seguida carregar seu programa clicando em “Enviar usando programador” (Seta no topo esquerdo).

Para que o programa seja gravado na memória flash, os jumpers de BOOT0 e BOOT1 devem estar ambos em 0.

**GPIO’s:**

Sua programação, para pinos digitais, é a mesma que o arduino, diferenciando-se apenas na forma de se referir aos pinos. Ao invés de números usa-se o nome que está escrito na placa precedido de P. por exemplo se vou manipular o pino A13, no código me refiro a ele como PA13. Para mais informações sobre o código em si, confira a seção arduino.

Todos os pinos do STM32 podem ser usados como pinos digitais, lembrando que estes trabalham lendo e fornecendo tensões de 3,3V. Introduzindo qualquer valor acima disso pode danificá-los. A corrente máxima que cada pino pode fornecer é de 20mA.

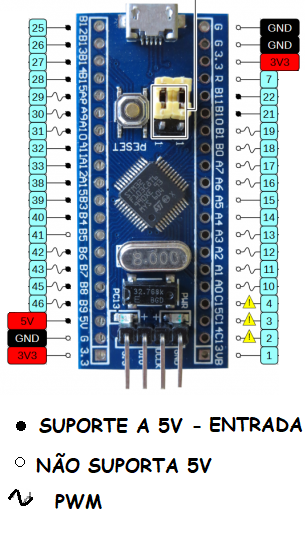
Pinos especiais:

Alguns pinos suportam PWM, eles estão apresentados na figura a seguir. Além disso, a programação de um sinal PWM pode ser feita da mesma forma que é para o arduino.

Os pinos PC13, PC14 e PC15 são pinos que recomendo não estarem conectados a nenhum periférico externo.

No pino PC13 está um led embutido na placa que pode ser usado na programação.

Os pinos marcados abaixo podem suportar uma entrada de 5V, desde que estes sejam configurados como **entrada**:



**Encoder Interface:**

O stm32 possui em seus timers um modo de operação dedicado ao uso de encoders incrementais. Usando esse modo a contagem do encoder é feita pelo hardware, incrementando ou decrementando diretamente o valor de um Timer, sem a intervenção da CPU, o que otimiza esse processo e libera a CPU para outras tarefas.

Nesse modo, as conexões A e B do encoder são conectadas cada uma em um canal de um dado Timer. Ao gerar pulsos, elas funcionarão como sinal de clock para o Timer, no entanto há um circuito interno que, com base na diferença de fase dos sinais, define se o Timer incrementa ou decrementa, indicando assim a posição do encoder.

Na blue pill, apenas os Timers TIM2, TIM3 e TIM4 possuem esse recurso, sendo necessário conectar o encoder nos canais 1 e 2 deles, os quais estão relacionados de acordo com a tabela abaixo:

|  | **Canal 1** (CH\_1) | **Canal 2** (CH\_2) |
| --- | --- | --- |
| **TIM2** | PA0 | PA1 |
| **TIM3** | PA6 | PA7 |
| **TIM4** | PB6 | PB7 |

A seguir será abordado como é possível configurar os Timers, bem como ler os valores e até como zera-los usando o Arduino IDE:

Primeiro, devemos incluir a seguinte biblioteca, ela possui funções que facilitam o uso dos Timers:

#include <libmaple/timer.h>



Abaixo temos uma parte do código, que deve estar no void setup, para configurar o Timer TIM2 no modo Encoder Interface:

timer\_pause(TIMER2); // para a contagem do timer 2, desabilitando-o

timer\_set\_mode(TIMER2, TIMER\_CH1, TIMER\_ENCODER); // altera o modo do canal

timer\_set\_mode(TIMER2, TIMER\_CH2, TIMER\_ENCODER); // altera o modo do canal

timer\_generate\_update(TIMER2); // aplica a mudança nas configurações

timer\_resume(TIMER2); // reativa o timer 2



Para ler o valor salvo, basta chamar a seguinte função, e ela retornará um número inteiro de 16 bits com sinal:

int16\_t encoderTIM2 = timer\_get\_count(TIMER2); // Lê o contador do TIM2



Para reiniciar a contagem, basta alterar o valor do Timer para zero usando a seguinte função:

timer\_set\_count(TIMER2, 0); // zera a contagem do TIM2

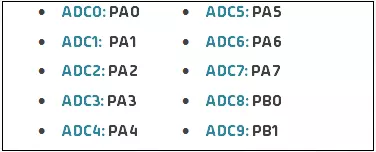


Para os Timer TIM3 e TIM4 basta trocar TIMER2 por TIMER3 ou TIMER4.

**Leitura analógica:**

Diferentemente do arduino, o stm32 opera em 3.3V e, portanto, suas entradas analógicas se limitam a, no máximo, 3.3V, sendo que se for introduzido um maior valor pode-se danificar a placa. Na blue pill a resolução do ADC é de 12 bits, retornando valores entre 0 e 4095. Já na programação, usa-se ‘analogRead’ para ler o valor de um pino analógico assim como é no arduino.

A blue pill possui 10 pinos analogicos, sendo eles apresentados na tabela abaixo:



Exemplo de leitura:

int leitura = analogRead(PA4); // ler o ADC4 do pino PA4

**EXTI:**

Se assemelha ao PCI do arduino, diferenciando-se fortemente pela forma como é expresso no código. Ao sofrer uma mudança de nível, um pino dispara uma função se configurado para isso, graças ao EXTI.

Assim como no arduino, alguns pinos compartilham da mesma função, sendo eles agrupados da seguinte maneira: PA0 e PB0 estão ligados a EXTI0, PA5 e PB5 estão ligados a EXTI5, e assim por diante. No total há 16 EXTI na blue pill.

No código, no arduino IDE, podemos usar quase a mesma sintaxe das interrupções externas do arduino:

attachInterrupt(pino, ISR, modo);

Onde:

* *pino* é o número do pino no qual a interrupção externa está entrelaçada.
* *ISR* é o nome da função que será disparada ao se atingir a condição especificada.
* *modo* é o tipo de condição que dispara a interrupção. Eles podem ser:
  + **RISING** : Aciona a interrupção quando o estado do pino for de LOW para HIGH apenas.
  + **FALLING** acionar a interrupção quando o estado do pino for de HIGH para LOW apenas.

O pino utilizado deve ser previamente definido como ENTRADA.

**Sensores**

**HC-SR04:**

Ele é um sensor de distância capaz de medir distâncias entre 2cm e 4m. É muito utilizado na robótica devido ao seu baixo preço e a facilidade de aplicação. Este sensor funciona com ultrassom. Quando solicitado ele dispara um pulso ultrassônico que é lançado, bate no objeto a ser detectado, depois retorna a ele que indica que o som foi retornado.

Esse é seu aspecto físico e seus terminais:



Os terminais **VCC** e **GND** são a alimentação do sensor, que deve funcionar com 5V. **Trig** e **Echo** são, respectivamente, o Gatilho, que serve para indicar ao sensor que este possa disparar uma onda ultrassônica, e a Resposta, que permanece em nível alto enquanto a onda não retorna.

Para utilizá-lo devemos conectar o Echo a uma entrada digital do microcontrolador e o TRIG a uma saída digital. A ideia que garante o seu funcionamento deve ser a seguinte:

* Deve-se gerar um pulso de pelo menos 10us no pino do TRIG. Ao fazer isso, o sensor envia um pulso ultrassônico e faz o pino do ECHO ir para nível **ALTO**.
* Após receber o ultrassom de volta, o sensor faz com que o ECHO volte para o nível **BAIXO**.
* O microcontrolador deve medir o tempo em que o ECHO ficou em nível alto. Esse será o tempo de ida e volta do ultrassom, sabendo isso e a velocidade do som no ar, podemos calcular a distância usando a fórmula:

Abaixo segue um exemplo de uso com o arduino:

#define ECHO 2

#define TRIG 3

#define VEL\_SOM 0.0343 // em cm/us

void setup(){

pinMode(ECHO, INPUT); // Echo como entrada

pinMode(TRIG, OUTPUT); // Trig como saída

Serial.begin(9600); // habilita o monitor serial

}

void loop(){

digitalWrite(TRIG,1); // nível alto, inicio do pulso

delayMicroseconds(10); // espera 10us, duração do pulso

digitalWrite(TRIG, 0); // fim do pulso

// retorna quanto tempo, em us, ECHO ficou em nível alto

long int tempo = pulseIn(ECHO, 1); // pulseIn(pino, nível)

float distancia = (VEL\_SOM\*tempo)/2; //divide por dois, ida e volta

Serial.print(distancia);

Serial.println("cm");

}





**Coletando dados**



**REFERÊNCIAS**

*Instalação de Pacotes Debian :* [*https://help.ubuntu.com/kubuntu/desktopguide/pt\_BR/manual-install.html*](https://help.ubuntu.com/kubuntu/desktopguide/pt_BR/manual-install.html)

*Instalando o FTP:*

* vsftpd:[*https://greenhost.cloud/how-to-set-up-vsftpd-on-ubuntu-24-04/*](https://greenhost.cloud/how-to-set-up-vsftpd-on-ubuntu-24-04/)
* *vscode:* [*https://king.host/wiki/artigo/como-conectar-ftp-com-visual-studio-code/*](https://king.host/wiki/artigo/como-conectar-ftp-com-visual-studio-code/)

Python:

* threading: <https://www.dio.me/articles/executando-tarefas-simultaneas-em-python-com-a-biblioteca-threading>
* rich: [https://rich-readthedocs-io.translate.goog/en/stable/introduction.html](https://rich-readthedocs-io.translate.goog/en/stable/introduction.html?_x_tr_sl=en&_x_tr_tl=pt&_x_tr_hl=pt&_x_tr_pto=tc)
* pySerial: <https://pyserial.readthedocs.io/en/latest/>
* subprocess: <https://pydoc-zh.readthedocs.io/en/latest/library/subprocess.html>
* socket: <https://docs.python.org/3/library/socket.html>

*GPIOs:*

* *Instalação e saídas:* [*https://ubuntu.com/tutorials/gpio-on-raspberry-pi#2-installing-gpio*](https://ubuntu.com/tutorials/gpio-on-raspberry-pi#2-installing-gpio)
* *Entradas:* [*https://www.makerhero.com/guia/raspberry-pi/botoes-raspberry-pi/?srsltid=AfmBOopYnGT6VRHlWUN-omDFMUqdnoNFP9\_5aAfkIySW9V4utcM8p\_JN*](https://www.makerhero.com/guia/raspberry-pi/botoes-raspberry-pi/?srsltid=AfmBOopYnGT6VRHlWUN-omDFMUqdnoNFP9_5aAfkIySW9V4utcM8p_JN)

*LCD:*

* *Conexão:* [*https://medium.com/@thedyslexiccoder/how-to-set-up-a-raspberry-pi-4-with-lcd-display-using-i2c-backpack-189a0760ae15*](https://medium.com/@thedyslexiccoder/how-to-set-up-a-raspberry-pi-4-with-lcd-display-using-i2c-backpack-189a0760ae15)
* *Driver, funções básicas e smbus:* [*https://www.arduinoecia.com.br/como-usar-display-lcd-i2c-raspberry-pi/?srsltid=AfmBOoqgLyqUnrzisYerYokUv1F4QC5IRO8K05X2UQFwbV7ou9q86MQa*](https://www.arduinoecia.com.br/como-usar-display-lcd-i2c-raspberry-pi/?srsltid=AfmBOoqgLyqUnrzisYerYokUv1F4QC5IRO8K05X2UQFwbV7ou9q86MQa)
* *i2cdetect:* [*https://ubuntu.com/tutorials/gpio-on-raspberry-pi#4-i2c-example*](https://ubuntu.com/tutorials/gpio-on-raspberry-pi#4-i2c-example)

*Arduino:*

* *Timers:* [*https://embarcados.com.br/timers-do-atmega328-no-arduino/*](https://embarcados.com.br/timers-do-atmega328-no-arduino/)
* *Interrupções externas:* [*https://reference.arduino.cc/reference/pt/language/functions/external-interrupts/attachinterrupt/*](https://reference.arduino.cc/reference/pt/language/functions/external-interrupts/attachinterrupt/)
* *Pin Change Interrupt:* [*https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P\_Datasheet.pdf*](https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf)

*STM32:*

* *Introdução:* [*https://embarcados.com.br/ja-ouviu-falar-sobre-stm32/*](https://embarcados.com.br/ja-ouviu-falar-sobre-stm32/)
* *Programação:* [*https://autocorerobotica.blog.br/crie-projetos-com-stm32-utilizando-a-ide-arduino/*](https://autocorerobotica.blog.br/crie-projetos-com-stm32-utilizando-a-ide-arduino/)
* *GPIO’s:* [*https://curtocircuito.com.br/blog/arm/conhecendo-o-bluepill-stm32*](https://curtocircuito.com.br/blog/arm/conhecendo-o-bluepill-stm32)
* *Encoder Interface:*
  + [*https://www.st.com/resource/en/datasheet/cd00161566.pdf*](https://www.st.com/resource/en/datasheet/cd00161566.pdf)
  + [*https://deepbluembedded.com/stm32-timer-encoder-mode-stm32-rotary-encoder-interfacing/*](https://deepbluembedded.com/stm32-timer-encoder-mode-stm32-rotary-encoder-interfacing/)
* *Leitura Analógica:* [*https://curtocircuito.com.br/blog/post/conhecendo-o-bluepill-stm32?srsltid=AfmBOorjBanUxkHddLUwZhoKcUOpUYwxa-hCQWrwqy-H\_Ld8qlsERMO6*](https://curtocircuito.com.br/blog/post/conhecendo-o-bluepill-stm32?srsltid=AfmBOorjBanUxkHddLUwZhoKcUOpUYwxa-hCQWrwqy-H_Ld8qlsERMO6)
* *EXTI:*