

ARDUINO

Explorando o Arduino

Projeto UERJ - Faperj

Professor: Austeclynio Pereira - 2023

Explorando o Arduino



- Início em 10/01/2023;
- Término em 23/02/2023;
- Horário: das 9h às 12h;
- Dias da semana: 3as e 5as;
- Conteúdo do curso em:
www.inovuerj.sr2.uerj.br/portal/cursos_integra
- Teste ao final do curso;

Bibliografia



1. *Arduino Básico* – McRoberts, Michael. Novatec, 2015
2. *Exploring Arduino* – Blum, Jeremy. Wiley, 2016(existe em português)
3. *Building Internet of Things With the Arduino* – Doukas, Charalampos. CreateSpace Publishing, 2012
4. *Make: Electronics* – Platt, Charles. Maker Media, 2016(existe em português)
5. *Programming Arduino: Getting Started with Sketches* – Monk, Simon. McGraw-Hill Education, 2016
6. *Movimento, luz e som com Arduino e Raspberry Pi* – Monk, Simon. Novatec, 2016
7. <https://www.arduino.cc/>

Introdução



- Plataforma aberta de *hardware* orientada para prototipagem com microcontroladores combinada com uma IDE, para programá-lo;
- Surgiu, em 2005, no Interaction Design Institute de Ivrea, Itália;
- Tinha o propósito de ser um ferramenta de uso fácil, voltada para estudantes sem formação em eletrônica e programação;
- O *software* é derivado de um projeto de código aberto, denominado Wiring, criado por um aluno do instituto;
- Existem IDEs tanto para a linguagem C quanto para **MicroPython**. Rodam em Linux, Windows e Mac;

Introdução



- Não executa nenhum Sistema Operacional;
- Pode ser utilizado em projetos autosuficientes ou conectado em um computador de mesa;
- Pode ser conectado a uma rede local ou à web, estimulando projetos em IoT;
- Sua popularidade deve-se a:
 - baixo custo(Arduino Uno < US\$4.00 + frete);
 - *open-source*;
 - fácil uso;
 - imensa quantidade de *hardware* de expansão(sensores, atuadores e *shields*);
- Apresentado em diferentes formas e tamanhos;

Introdução



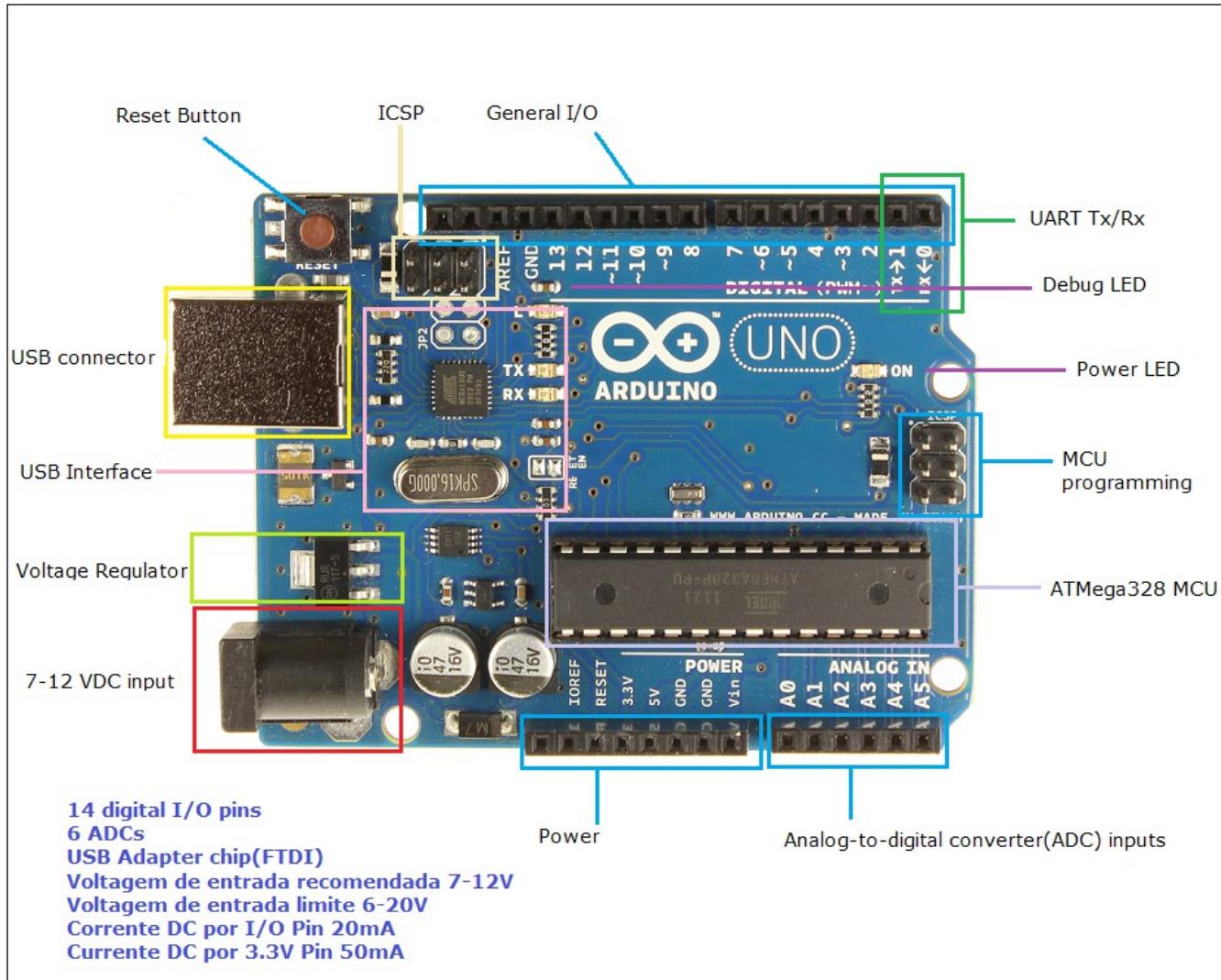
- Por ser uma plataforma aberta qualquer um pode projetar uma nova versão do Arduino e distribuí-la;
- Estima-se que, entre placas oficiais e seus clones, já tenham sido comercializadas mais de 2.000.000 placas;
- Algumas placas Arduino não possuem um conversor USB nativo, então é necessário uma placa adaptadora USB;
- Para construir uma placa são necessários apenas 3 componentes:
 - uma MCU Atmega168/Atmega328;
 - um oscilador de 16Mhz;
 - uma fonte de +5V;

Introdução



- Há 3 tipos de interfaces com o Arduino;
 - Digital
 - Analógica
 - Protocolos seriais de comunicação:
 - TTL serial
 - I²C
 - 1-Wire
 - Serial Peripheral Interface(SPI)

A Anatomia do Arduino



A MCU Atmel



- Um microcontrolador, em essência, é um computador em um único *chip*;
- Inclui:
 - CPU;
 - ROM;
 - RAM;
 - portas de comunicação serial;
 - conversores de sinais etc;
- É chamado de microcontrolador devido a sua pequena dimensão e capacidade de controlar máquinas e engenhocas;

A MCU Atmel



- Grande parte das placas Arduino utilizam a MCU Atmel;
- Versões iniciais do Arduino usavam a MCU Atmel168;
- Um programa(*sketch*) no Arduino fica armazenado na memória *flash*;
- Os dados manuseados pelo programa ficam na memória SRAM;
- Dados de armazenamento não volátil ficam na EEPROM;
- Existe uma IDE, Atmel Studio, voltada para as MCUs Atmel;
- Somente em versão para Windows;

A MCU Atmel - modelos



MCU	clock(*)	flash	SRAM	EEPROM
ATmega328	16MHz	32k	2k	1k
ATmega168	16Mhz	16k	1k	512
ATmega2560	16Mhz	256k	8k	4k
ATmega32U4	16Mhz	32k	2.5k	1k
ARM Cortex	84Mhz	512k	96k	0

(*) Com voltagem de entrada >=5v

Placas Arduino



Arduino ProMini

Necessário um FTDI Adapter

Sem DC power jack

14 digital I/O pins

6 ADCs

1 UART (Universal Asynchronous Receiver/Transmitter)

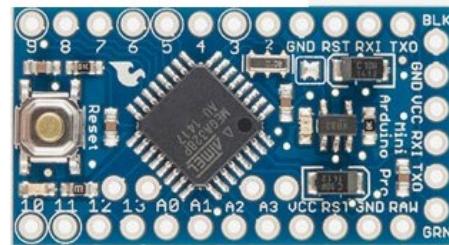
Voltagem de operação 3.3V or 5V (depende do modelo)

Voltagem de entrada 3.35 -12 V (3.3V modelo) or 5 - 12 V (5V modelo)

Corrente DC por I/O pin 40mA

Tamanho 33 x 18 mm

MCU ATmega168 ou Atmega328



Arduino NANO

Sem DC power jack

14 digital I/O pins

8 ADCs

1 UART

Voltagem de operação 5V

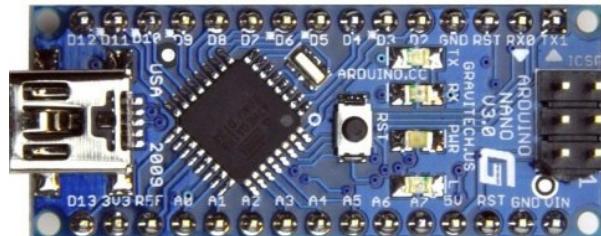
Voltagem de entrada recomendada 7-12V

Corrente DC por I/O pin 40mA

Tamanho 45 x 18 mm

MCU Atmega328

Ideal para projetos prontos

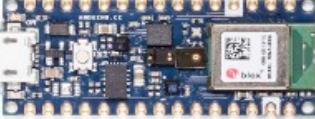


Família Arduino NANO



Nano Family

The Nano Family is a set of boards with a tiny footprint, packed with features. It ranges from the inexpensive, basic Nano Every, to the more feature-packed Nano 33 BLE Sense / Nano RP2040 Connect that has Bluetooth® / Wi-Fi radio modules. These boards also have a set of embedded sensors, such as temperature/humidity, pressure, gesture, microphone and more. They can also be programmed with MicroPython and supports Machine Learning.

			
Arduino Nano 33 IoT	Arduino Nano RP2040 Connect	Arduino Nano 33 BLE Sense	
			
Arduino Nano 33 BLE	Arduino Nano Every	Arduino Nano	Arduino Nano Motor Carrier

Placas Arduino



Arduino Mega

54 digital I/O pins

16 ADCs

4 UARTS

Voltagem de operação 5V

Voltagem de entrada recomendada 7-12V

Voltagem de entrada limite 6-20V

Corrente DC por I/O Pin 20mA

Corrente DC por 3.3V Pin 50mA

Tamanho 101 x 53 mm

MCU Atmega2560



Arduino Mega ADK

54 digital I/O pins

16 ADCs

4 UARTS

Capacidade de ser um hospedeiro USB

Voltagem de operação 5V

Voltagem de entrada recomendada 7-12V

Voltagem de entrada limite 6-20V

Corrente DC por I/O Pin 40mA

Corrente DC por 3.3V Pin 50mA

Tamanho 101 x 53 mm

MCU ATmega2560



Placas Arduino



Arduino Due

54 digital I/O pins

12 ADCs

4 UARTS

2 DACs (saídas senoidas). Bom para áudio

Capacidade de ser um hospedeiro USB

Voltagem de operação 3.3V

Voltagem de entrada recomendada 7-12V

Voltagem de entrada limite 6-16V

Corrente DC por I/O Pin 130mA

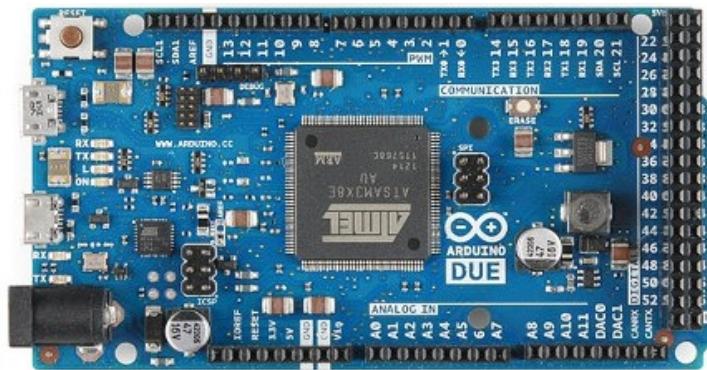
Corrente DC por 3.3V Pin 800mA

Corrente DC por 5V Pin 800mA

Tamanho 101 x 53 mm

MCU ARM Cortex. Não possui EEPROM

Ideal para projetos de larga escala



Arduino LyliPad USB

Para versões antigas necessário um FTDI Adapter

Orientado para projetos vestíveis

9 digital I/O pins

4 ADCs

1 UART

Voltagem de operação 3.3V

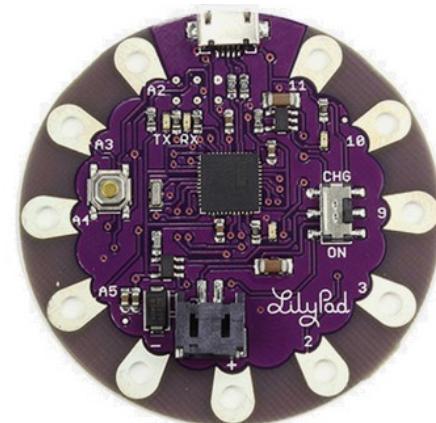
Voltagem de entrada recomendada 3.8-5V

Corrente DC por I/O pin 40mA

Tamanho 65 mm

A baixa voltagem de trabalho provoca redução do clock(8 Mhz)

MCU ATmega32u4



Placas Arduino



Arduino Yún

20 digital I/O pins

12 ADCs

1 UART

Como microprocessador Atheros AR9331 embutido

Suporte a Ethernet e Wi-Fi nativos, ideal para a IoT

Slot para cartão Micro-SD

Capacidade de ser um hospedeiro USB

Voltagem de operação 5V

Voltagem de entrada recomendada 5V

Corrente DC por I/O Pin 40mA

Corrente DC por 3.3V Pin 50mA

Tamanho 68 x 53 mm

MCU Atmega32u4 e Atheros AR9331

Suporte ao Linux (LininoOS)



Placas MKR



The MKR Family is a series of boards, shields & carriers that can be combined to create amazing projects without any additional circuitry. Each board is equipped with a radio module (except MKR Zero), that enables Wi-Fi, Bluetooth®, LoRa®, Sigfox, NB-IoT communication. All boards in the family are based on the [Cortex-M0 32-bit SAMD21](#) low power processor, and are equipped with a crypto chip for secure communication.

The MKR Family shields & carriers are designed to extend the functions of the board: such as environmental sensors, GPS, Ethernet, motor control and RGB matrix.

Boards

		
Arduino MKR 1000 WiFi	Arduino MKR WiFi 1010	Arduino MKR FOX 1200
		
Arduino MKR WAN 1300	Arduino MKR WAN 1310	Arduino MKR GSM 1400

Placas MKR(cont.)



Arduino MKR NB 1500



Arduino MKR Vidor 4000



Arduino MKR Zero

Placas Arduino

Portenta H7



- Nova família de placas Arduino orientada para IoT, IA e Machine Learning;
- O processador principal do H7 é o dual core STM32H747, incluindo um Cortex® M7 rodando a 480 MHz e um Cortex® M4 rodando a 240 Mhz;
- Os dois núcleos se comunicam por meio de um mecanismo de “chamada de procedimento remoto” que permite chamar funções no outro processador;
- Ambos os processadores compartilham todos os periféricos do chip e podem executar código de alto nível simultaneamente com tarefas em tempo real;
- É possível executar o código compilado do Arduino junto com o do MicroPython e ter os dois núcleos para se comunicarem entre si;
- Possibilidade de conectar um monitor externo para construir seu próprio computador embutido com uma interface de usuário;
- Possível graças à GPU integrada do processador STM32H747, o Chrom-ART Accelerator™;
- 8MB SDRAM;
- 16MB NOR Flash;
- 10/100 Ethernet ;
- USB HS;
- NXP SE050C2 Criptografia;
- WiFi/BT Módulo;
- Antena Externa;
- DisplayPort over USB-C;
- Adiciona dois conectores de alta densidade de 80 pinos na parte inferior da placa;

Placas Arduino

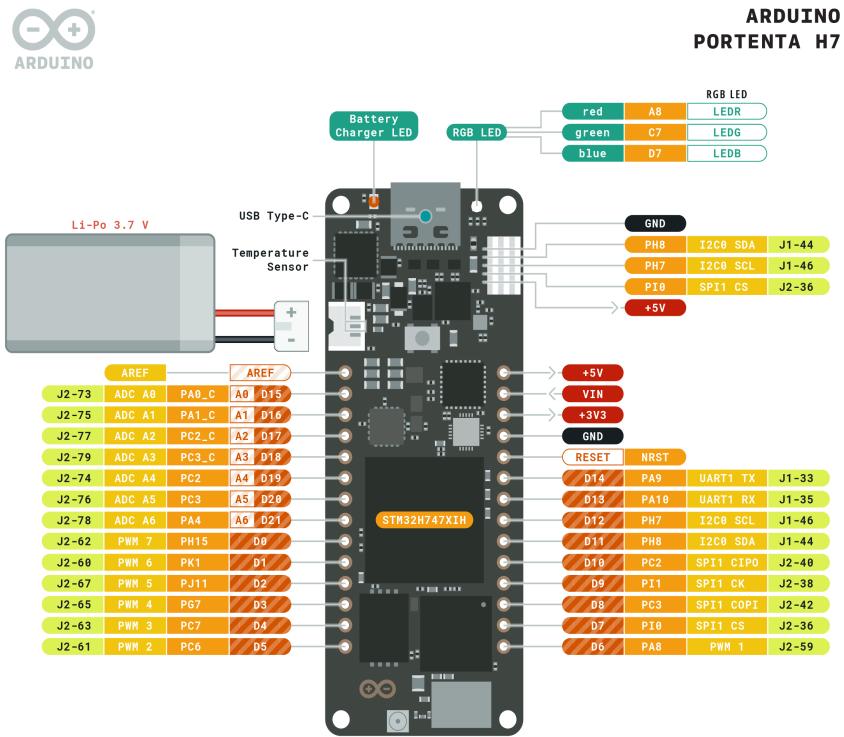
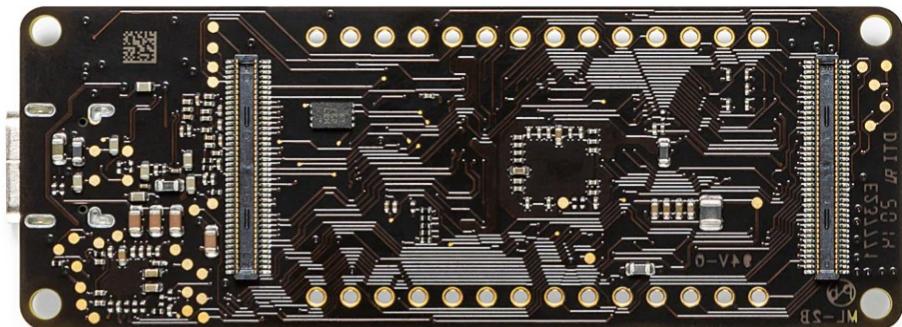
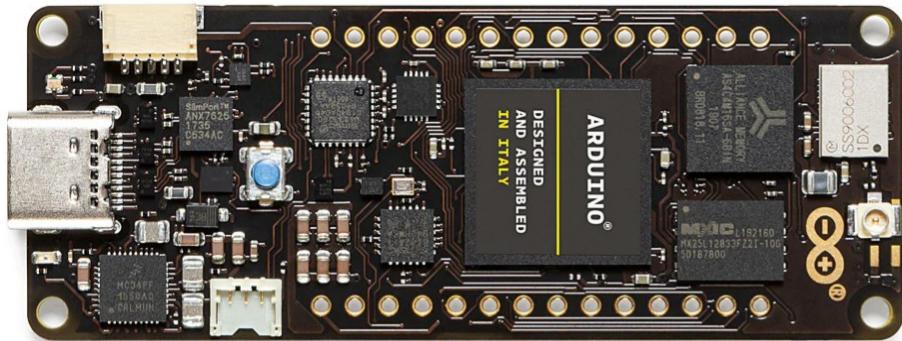
Portenta H7



- Radio module Murata 1DX dual WiFi 802.11b/g/n 65 Mbps and Bluetooth® (Bluetooth® Low Energy. 5 via Cordio stack, Bluetooth® Low Energy 4.2 via Arduino Stack);
- Secure Element NXP SE0502;
- Board Power Supply (USB/VIN) 5V;
- Supported Battery Li-Po Single Cell, 3.7V, 700mAh Minimum (integrated charger);
- Circuit Operating Voltage 3.3V;
- Display Connector MIPI DSI host & MIPI D-PHY to interface with low-pin count large display;
- GPU Chrom-ART graphical hardware Accelerator™;
- Timers 22x timers and watchdogs;
- UART 4x ports (2 with flow control);
- Ethernet PHY 10 / 100 Mbps;
- SD Card Interface for SD Card connector;
- Operational Temperature -40 °C to +85 °C;
- High-density Connectors Two 80 pin connectors will expose all of the board's peripherals to other devices;
- Camera Interface 8-bit, up to 80 Mhz;
- ADC 3x ADCs with 16-bit max. resolution (up to 36 channels, up to 3.6 MSPS);
- DAC 2x 12-bit DAC (1 MHz)
- USB-C Host / Device, DisplayPort out, High / Full Speed, Power delivery;
- **US\$ 114.00;**

Placas Arduino

Portenta H7



■ Ground ■ Internal Pin ■ Digital Pin ■ Microcontroller's Port
■ Power ■ SWD Pin ■ Analog Pin ■ High Density Connector
■ LED ■ Other Pin ■ Default

ARDUINO .CC

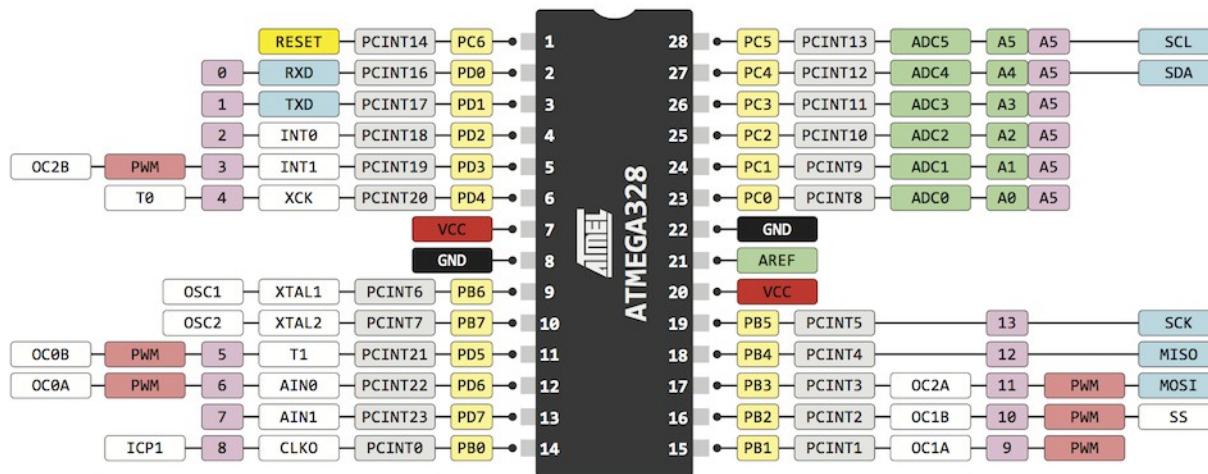
This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94039, USA.

MCU ATmega328



THE
DEFINITIVE
ATMEGA328
&Arduino
PINOUT DIAGRAM

GND
Power
Control
Physical Pin
Port Pin
Pin Function
Digital Pin
Analog Related Pin
PWM Pin
Serial Pin
IDE



www.pigpiod.com



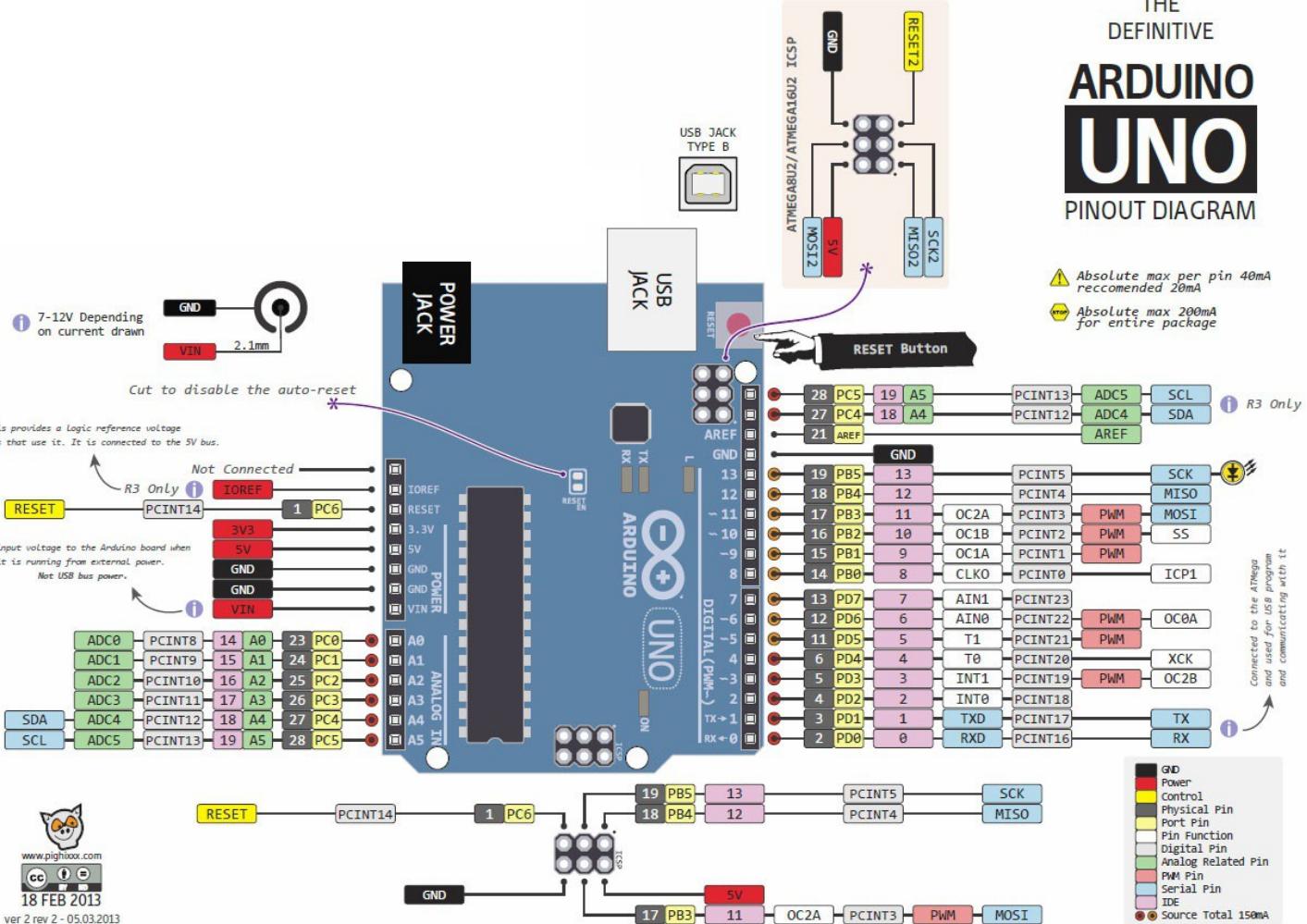
18 FEB 2013

ver 2 rev 0 - 19.02.2013

Arduino UNO



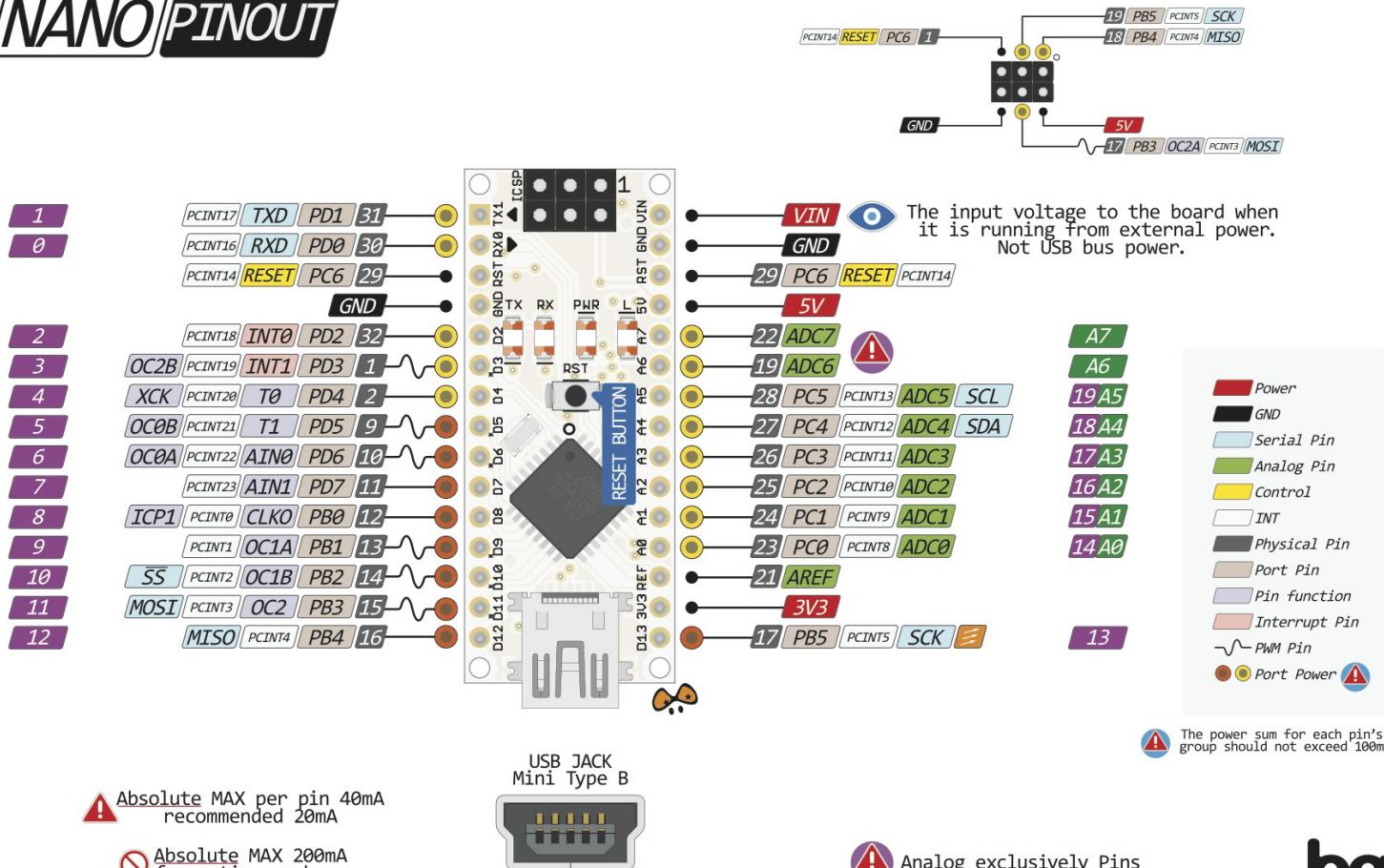
THE
DEFINITIVE
ARDUINO
UNO
PINOUT DIAGRAM



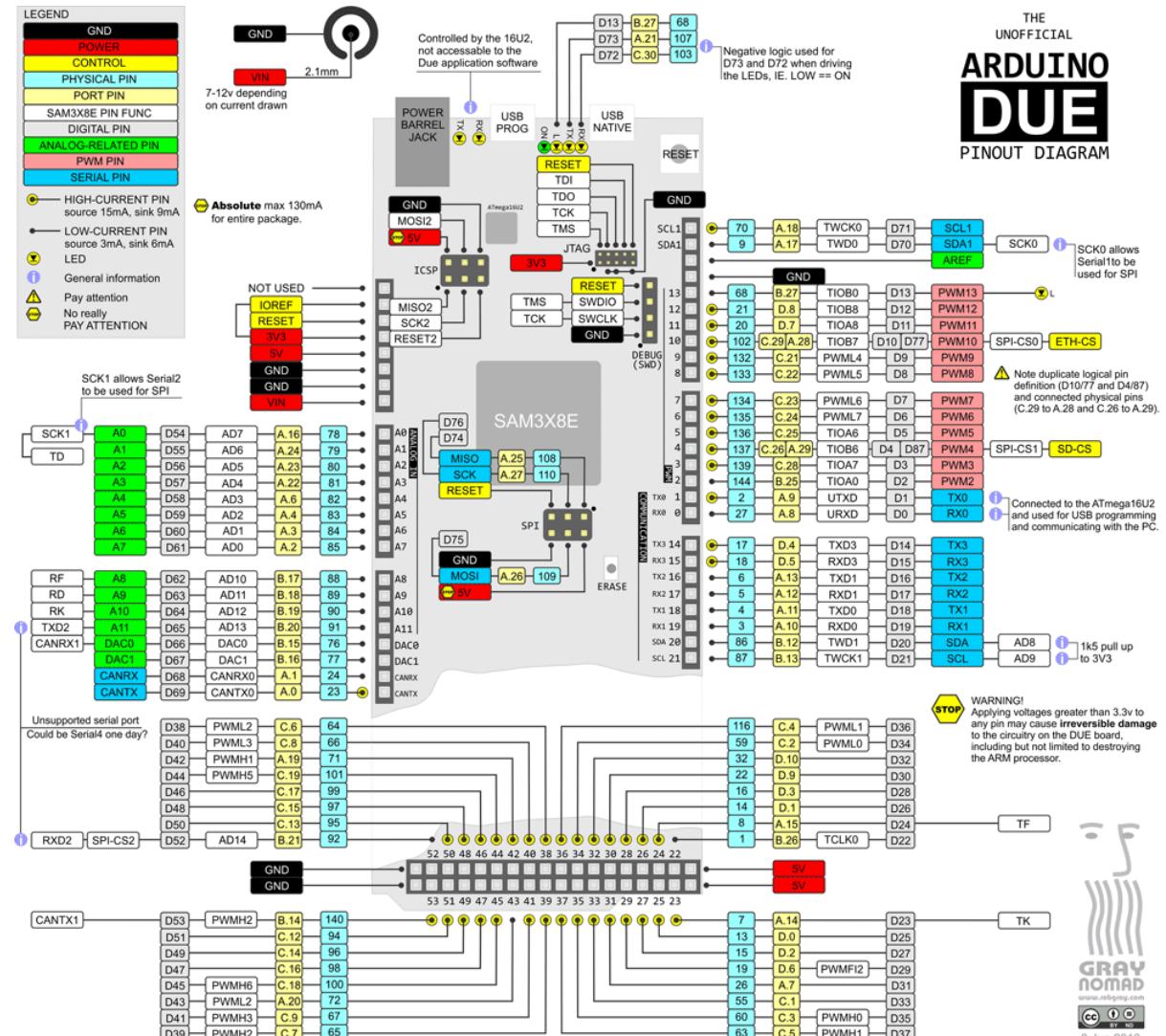
Arduino NANO



NANO PINOUT



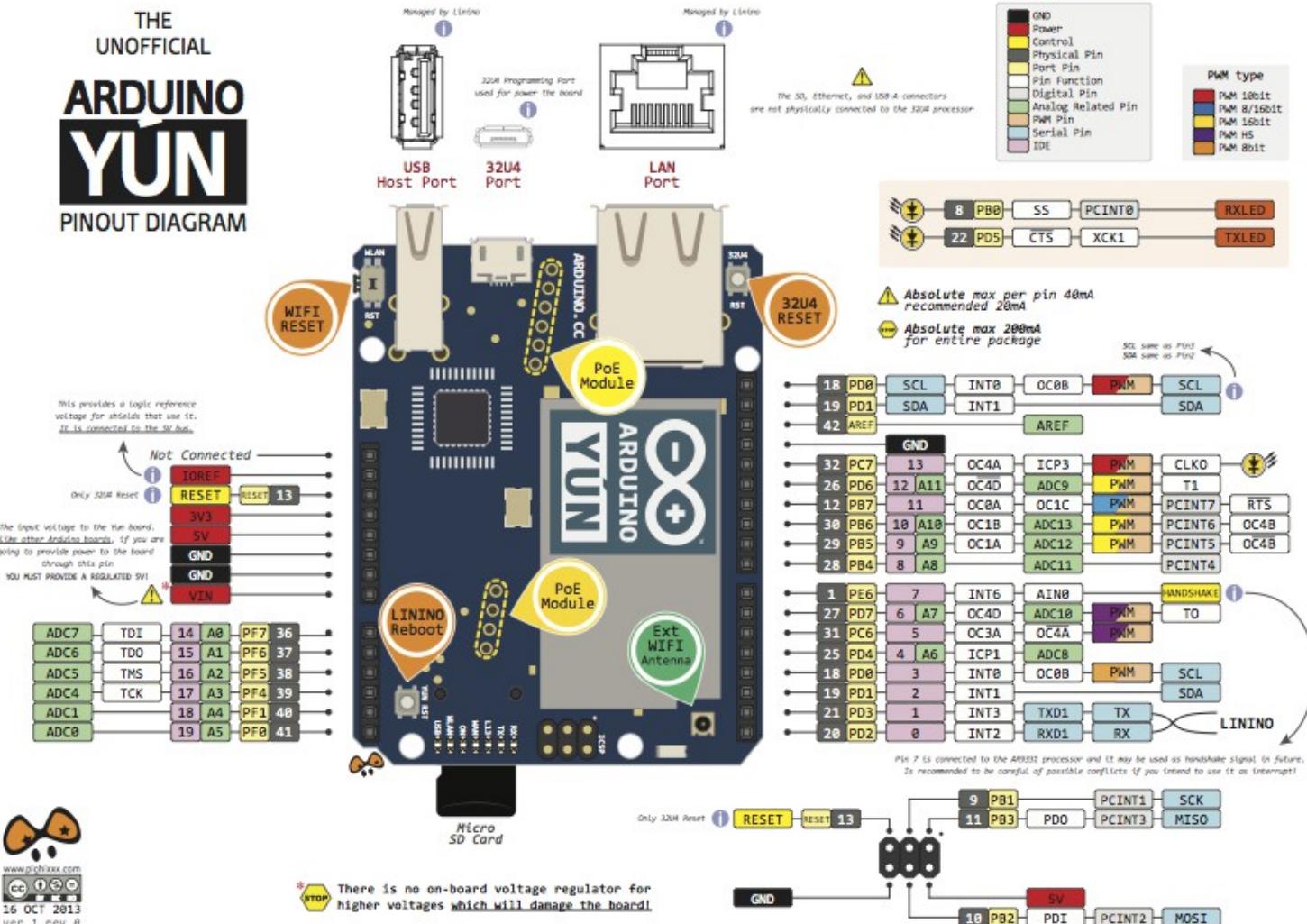
Arduino DUE



Arduino YÚN



THE
UNOFFICIAL
ARDUINO
YUN
PINOUT DIAGRAM



Sensores



- Têm a capacidade de poder antecipar uma demanda baseados em uma informação capturada em um contexto específico;
- Detecta, sente ou mede estímulos físicos tais como:
 - Temperatura;
 - Umidade;
 - Pressão;
 - Força;
 - Movimentos linear e angular;
 - Distância;
 - Luminosidade;
 - Vibração etc;
- Estes estímulos são transformados em sinais analógicos ou digitais;

Sensores



- Os sinais analógicos são convertidos para valores digitais;
- Primeira grande onda de sensores surgiu nos anos 80, através da indústria automobilística, atuando em:
 - Freios;
 - *Airbags*;
 - Emissão de gases etc;
- Segunda grande onda começou em 2007 com o advento dos *smartphones*;
- São categorizados por seu propósito primário;

Sensores



- Um GPS, apesar de ser um receptor de sinais de rádio, é categorizado como um sensor de localização;
- Um sensor, em um único encapsulamento, pode ter várias sensibilidades;
- Exemplos: acelerômetro e giroscópio;

Sensores



Categoria	O que faz?	Exemplos
Medida de posição	Reage a trocas de posições angulares ou lineares do sensor	Potenciometro, efeito Hall, gps
Movimento e proximidade	Reage a movimentos externos dentro da area de alcance do sensor	Proximidade ultrasonico, reflectivo ótico, infra-vermelho-passivo, proximidade capacitivo
Inercial	Reage a trocas no movimento físico do próprio sensor	Acelerometro, giroscópio, potenciometro, tilt
Pressão e força	Reage a força exercida sobre o sensor	Barometro, potenciometro de pressão
Óptico	Reage a presença de luz ou na mudança de sua intensidade	LDR, fotodiodos, fototransistor, sensores reflectivos, células solares
Imagen	Detecta uma imagem e a transforma em um sinal digital	CMOS Image Sensor
Magnético	Reage a presença de um campo magnético	Efeito Hall, chave magnética
Ambiente	Reage a presença de substâncias físicas no ambiente	Gases diversos, fumaça, umidade, poeira
Voltagem e corrente	Reage a trocas no fluxo de eletricidade em um circuito	Efeito Hall, amperimetro DC e AC, voltmetro DC e AC
Temperatura	Reage a quantidade de calor detectada	Termistor, termoacoplador, termometro infra-vermelho

Sensores

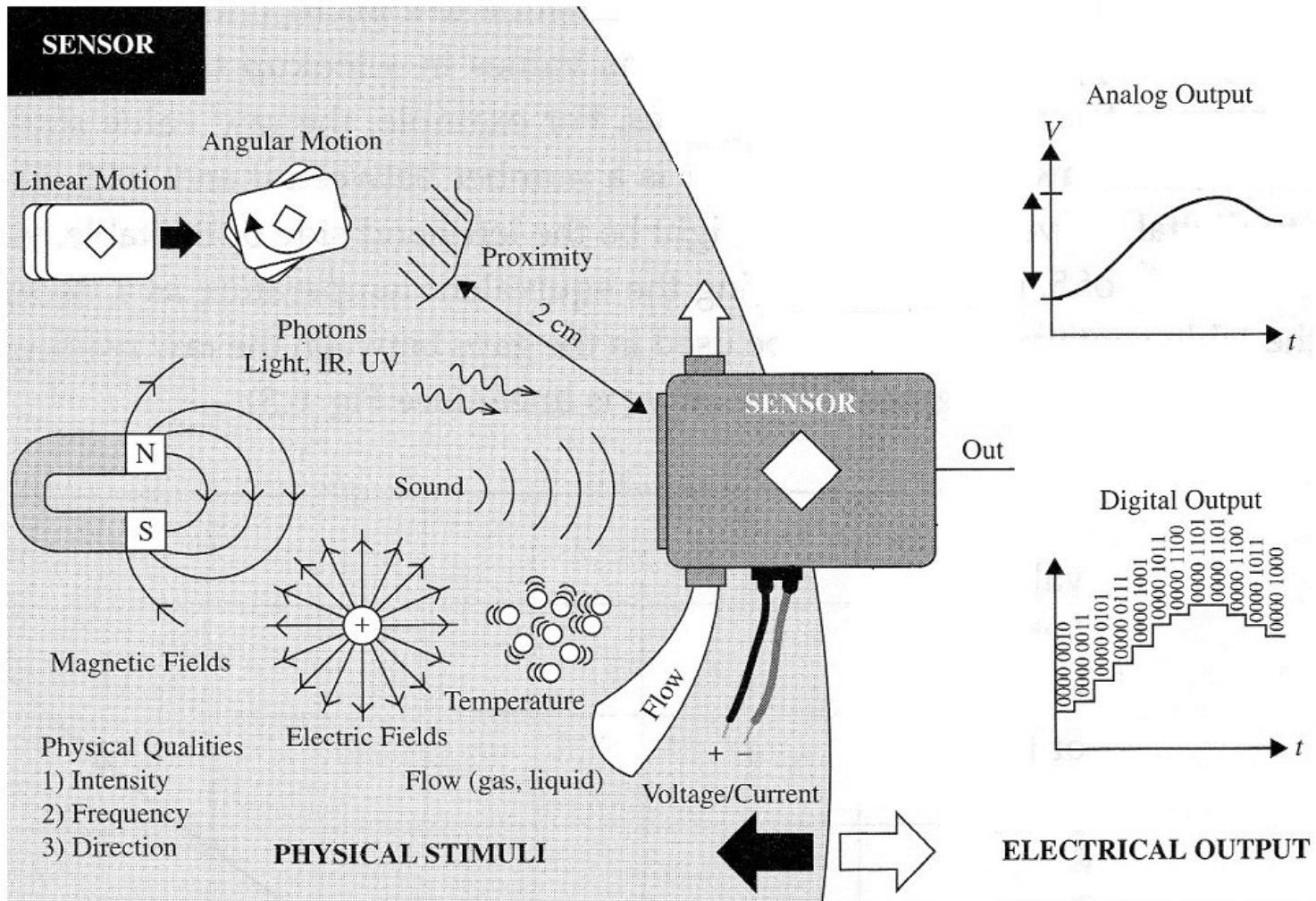
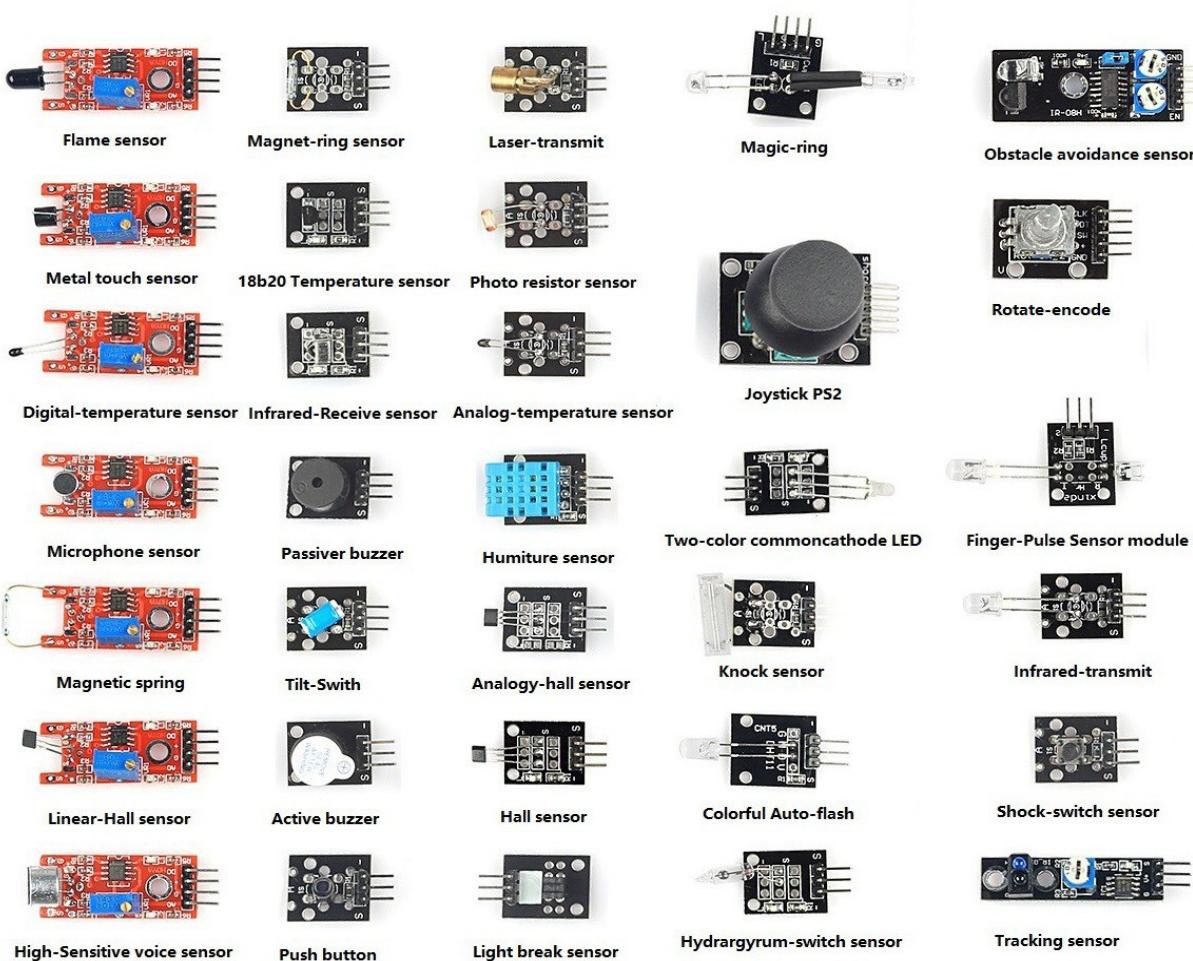


Figura extraída do livro Practical Electronics for Inventors

Sensores



Obs.: Códigos para testes em KITSENAT.

Atuadores



■ São dispositivos que transformam um sinal de entrada em um evento físico tais como:

- Movimento;
- Magnetismo;
- Som;
- Calor;

■ Requer um sinal de controle e uma fonte de energia;

■ O sinal de controle pode ser:

- Uma voltagem ou corrente elétricas;
- Uma pressão pneumática ou hidráulica;
- Uma intervenção humana;

Atuadores



■ A fonte de energia pode ser;

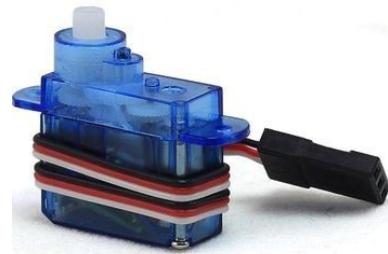
- Uma corrente elétrica;
- A pressão de um fluído hidráulico;
- Uma pressão pneumática;

■ Pode ser considerado um transdutor por ser capaz de transformar uma forma de energia em outra;

Atuadores



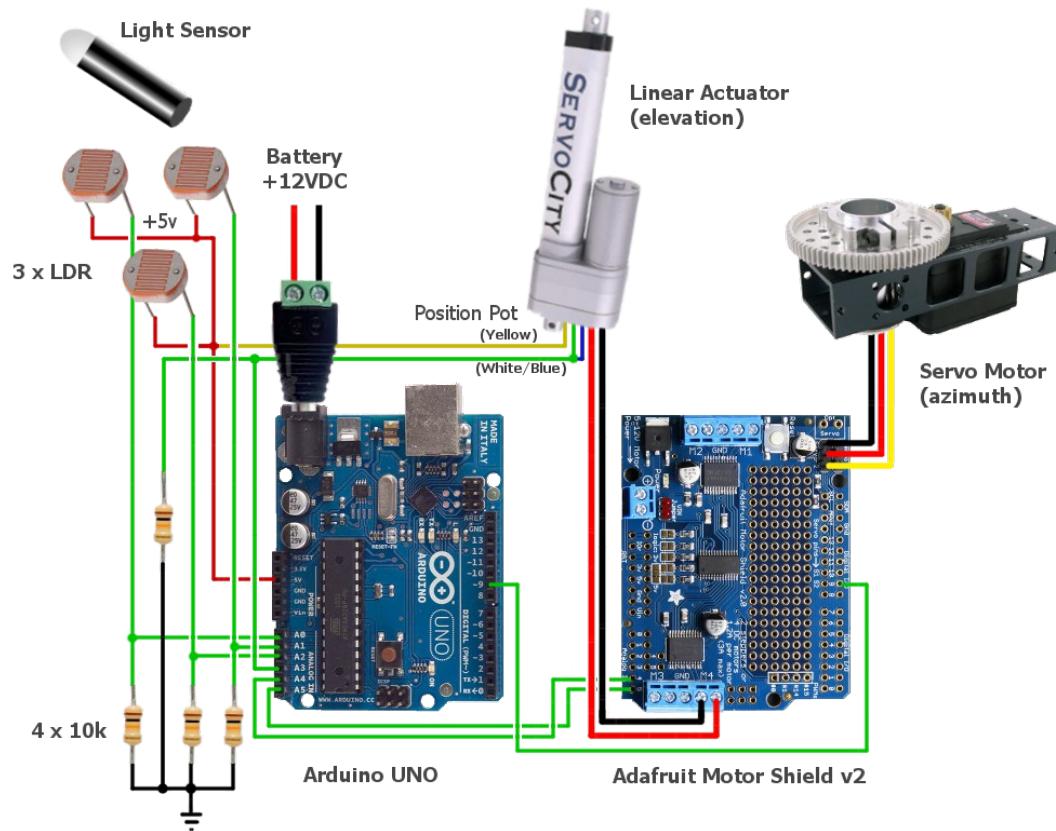
- Exemplos usados no Arduino: relé, motor DC, servo motor, stepper motor, buzzer;



Sensores e atuadores



Sun Tracking Solar Power Plant
Arduino / Motor Shield



Shields



- São dispositivos projetados para encaixarem-se diretamente nos soquetes de uma placa Arduino;
- Podem ser fomadas pilhas de *shields* em uma única placa Arduino;
- Neste caso devem ser verificados conflitos de pinagem;
- Maioria dos *shields* são orientados para o Arduino Uno, encontram-se alguns para o Arduino Mega;



Ethernet Shield



Motor Shield



Relay Shield

Eletrônica Básica



- Eletricidade, em geral, é uma forma de energia;
- Como a eletricidade não é usualmente vista, a não ser em relâmpagos, às vezes é difícil entendê-la;
- Ela faz um motor girar, acende lâmpadas, produz ondas de rádio, produz calor etc;
- É percebida quando existe um fluxo de elétrons em um elemento condutor;
- Este fluxo de elétrons, por unidade de tempo, é denominado corrente elétrica.
- A unidade de medida da corrente elétrica é o Ampére(A);

Eletrônica Básica

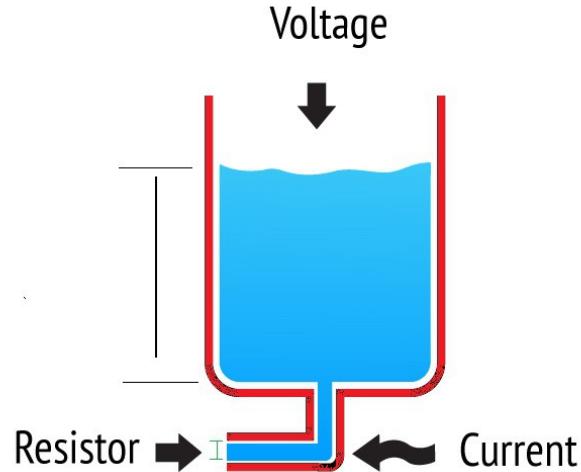
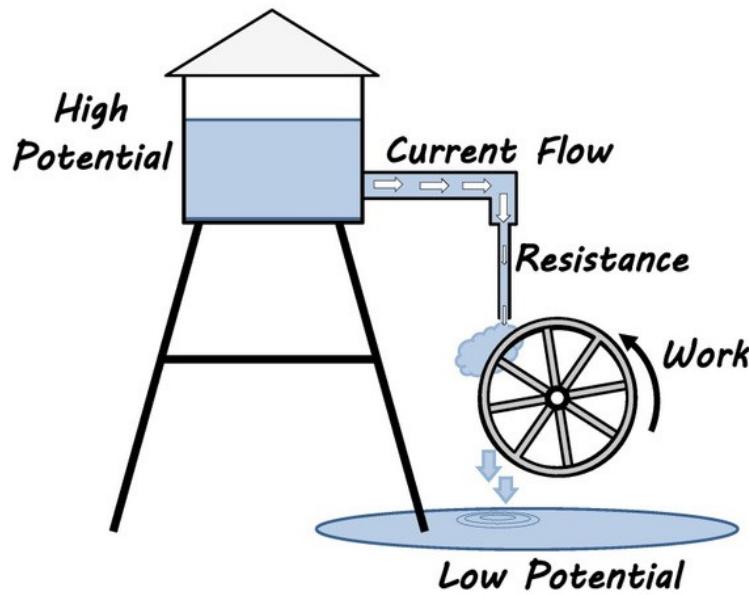


- Os elétrons são pressionados pelo condutor por uma força denominada força eletromotriz;
- A força eletromotriz surge em decorrência da aplicação de uma voltagem ao circuito;
- A unidade de medida da voltagem é o Volt(V);
- Agumas fontes de voltagem:
 - efeito eletroquímico;
 - efeito eletromagnético;
 - efeito fotovoltaico;
 - efeito termoelétrico;
 - efeito piezoelétrico;
 - efeito termonuclear;

Eletrônica Básica



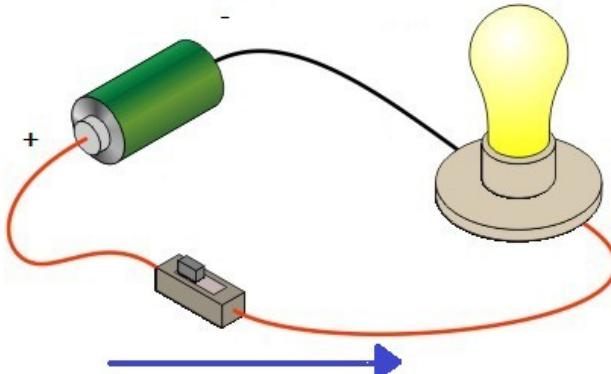
- Todo condutor possui uma resistência que se opõe ao fluxo dos elétrons;
- A unidade de medida da resistência é o Ohm(Ω);
- Esta resistência provoca a produção de calor no condutor;
- Abaixo, analogia com circuitos hidráulicos;



Eletrônica Básica



- Um sistema elétrico é chamado de circuito;
- Existem três tipos de circuitos: fechado, aberto e em curto;
- Para a corrente elétrica fluir é necessário que o circuito esteja fechado;
- Por convenção, a corrente elétrica sempre circula do polo positivo para o polo negativo, ou terra(GND);
- A corrente elétrica, como a água, sempre flui de um ponto de maior potencial de energia para um ponto de menor potencial;



Eletrônica Básica



- Ocorre trabalho quando uma pessoa, um animal ou uma máquina “empurra” algo para superar alguma resistência;
- Potência é a quantidade de trabalho realizado por unidade de tempo;
- A unidade de medida da potência é o Watt(W);
- O HP também é usado como unidade de medida de potência;
 - $1\text{ HP} = 740\text{ Watts}$
- Em termos elétricos, a potência de um circuito é o produto de sua voltagem pela corrente elétrica circulante;
 - $P = V \times I$

Eletrônica Básica

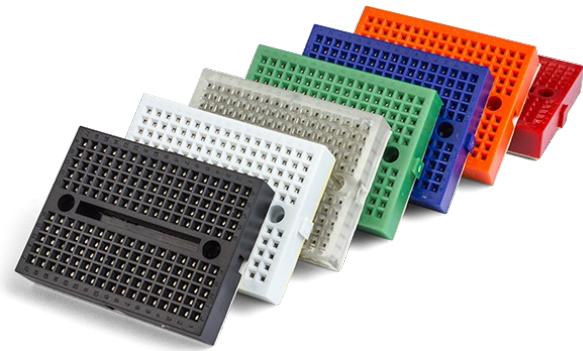
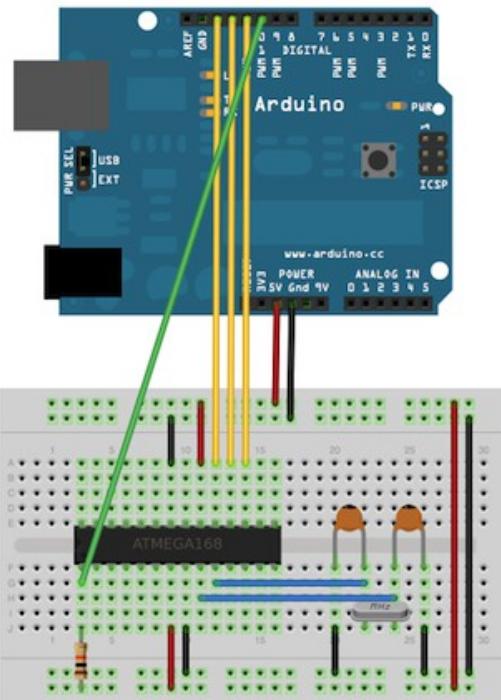
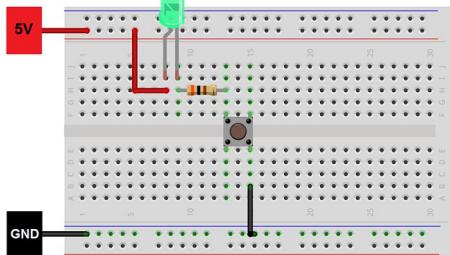


- Pela lei de Ohm, a resistência de um circuito é a razão entre a voltagem e a corrente circulante;
 - $R = V/I$
- Eletricidade deve sempre ser manuseada com cautela, mesmo no Arduino;
- Pode queimar a placa, componentes ou se machucar;

ProtoBoard



- Placa utilizada para a realização de experimentos;
- Ideal para a criação de protótipos;
- Dispensa o uso da soldagem dos componentes;



Ambiente de Desenvolvimento - IDE



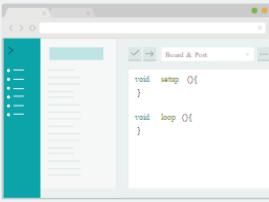
- O arduino executa um único programa(*sketch*), que precisa ser carregado em sua memória *flash*;
- A programação é realizada através da IDE do Arduino, localizada em www.arduino.cc/en/main/software;
- Existem duas versões de IDE: uma para uso *online*, outra para o *download*;
- Versões para *download*: 2.0(nova interface) e 1.8;

Access the Online IDE



ARDUINO WEB EDITOR
Start coding online with the [Arduino Web Editor](#), save your sketches in the cloud, and always have the most up-to-date version of the IDE, including all the contributed libraries and support for new Arduino boards. The Arduino Web Editor is one of the [Arduino Create platform's tools](#).

[Try It Now](#)
[Getting Started](#)



Download the Arduino IDE



ARDUINO 1.8.1
The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board. Refer to the [Getting Started](#) page for installation instructions.

[Windows Installer](#)
[Windows ZIP file for non admin install](#)

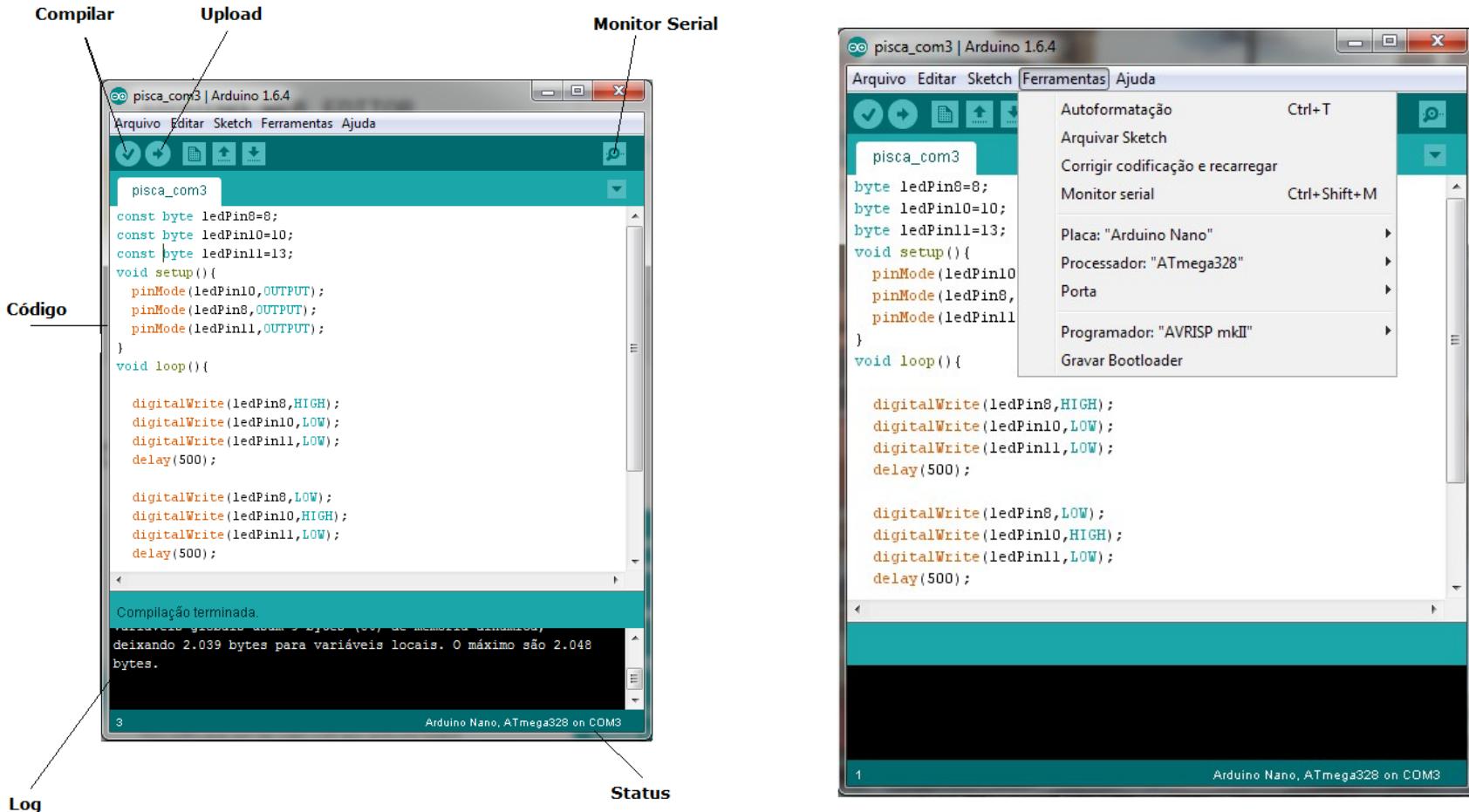
Windows app [Get](#)

[Mac OS X 10.7 Lion or newer](#)

[Linux 32 bits](#)
[Linux 64 bits](#)
[Linux ARM](#)

[Release Notes](#)
[Source Code](#)
[Checksums \(sha512\)](#)

Ambiente de Desenvolvimento - IDE



Vamos praticar! Usar a placa Arduino UNO.

Sketch



- Sketch é um conjunto de instruções logicamente organizadas com o propósito de realizar determinada tarefa;
- São codificados na IDE do Arduino onde lhe é atribuído, pelo programador, um nome;
- Este nome terá o sufixo **.ino**, automaticamente colocado pela IDE;
- Na pasta onde reside o sketch podem ser colocados outros arquivos de apoio, codificados em outro editor;
- Como existem placas com pinagem e MCUs diferentes, na IDE isto precisa ser definido;

Sketch



■ É constituído de:

- uma área destinada à definição de bibliotecas e dados;
- da função `setup()`;
- da função `loop()`

■ As funções **setup()** e **loop()** são de codificação obrigatória, mesmo que nada façam;

■ São sempre invocadas;

setup() e loop()



■ função setup()

- Aqui são codificadas instruções gerais tais como:
 - modo de operação dos pinos,
 - taxa de transmissão serial etc;
- Estas instruções são executadas apenas uma vez, sempre que o Arduino é energizado ou reiniciado;
- Pode chamar outras funções;

■ função loop()

- Aqui, as instruções codificadas são executadas repetidas vezes, sem fim;

Exemplo - Sketch



/* Created by David Cuartielles modified 30 Aug 2011 By Tom Igoe

This example code is in the public domain.

<http://www.arduino.cc/en/Tutorial/AnalogInput>*/

```
int sensorPin = A0; // select the input pin for the potentiometer
int ledPin = 13; // select the pin for the LED
int sensorValue = 0; // variable to store the value coming from the sensor

void setup() {
    pinMode(ledPin, OUTPUT); // declare the ledPin as an OUTPUT:
}

void loop() {
    sensorValue = analogRead(sensorPin); // read the value from the sensor:
    digitalWrite(ledPin, HIGH); // turn the ledPin on
    delay(sensorValue); // stop the program for <sensorValue> milliseconds:
    digitalWrite(ledPin, LOW) ; // turn the ledPin off:
    delay(sensorValue); // stop the program for for <sensorValue> milliseconds:
}
```

Tipos de dados, constantes e comentários



- Sketches, normalmente, manipulam dados;
- Dados, sejam constantes ou variáveis, precisam ser definidos quanto ao seu tipo;
- Dados podem ser de uso global ou local;
- Dados de uso global;
 - podem ser referenciados por todas as funções do *sketch*, são definidos no início do *sketch*;
- Dados locais;
 - São definidos no interior de uma função ou de um bloco de controle;
 - Em ambos os casos, são vistos apenas por eles;

Tipos de dados, constantes e comentários



- Dados declarados como estáticos têm seus valores preservados. Definidos pela palavra-chave *static*;
- Constantes são dados imutáveis. Definidas pela palavra-chave *const*;
- Sempre deve ser utilizada para definir um dado quando este não for passível de alteração;
- O nome de um dado pode ter até 32 caracteres;
- Deve começar sempre com uma letra ou com um _;
- Podem-se usar letras maiúsculas e minúsculas, números e _;

Tipos de dados, constantes e comentários



Tipo	bytes	Intervalo
boolean	1	true ou false
char	1	-128 a +128(Tabela ASCII)
byte	1	0 a 255
int	2	-32.768 a +32.768
unsigned int	2	0 a 65.536
long	4	-2.147.483.648 a + 2.147.483.647
unsigned long	4	0 a 4.294.967.295
float	4	-34028235E+38 a +34028235E+38
double	4	idem float

Tipos de dados, constantes e comentários



■ Fazendo declarações;

```
boolean chave;  
boolean chave=true;  
int valorInicial=0;  
char codigo='A';    ou    char codigo=65;  
byte numero = 8; ou byte numero=B1000;  
byte codigo='A';  
float valorInicial = 3456.678;  
const int=5;  
int valorEntrada, valorSaida;  
float valorEntrada=3.231,valorSaida;
```

Tipos de dados, constantes e comentários



- Devem ser atribuídos aos dados nomes que possam identificar com clareza o seu propósito;
- A mesma regra vale para os nomes dos *sketches* e para os nomes das funções;
- Comentário é uma informação colocada no *sketch* para descrever o que o código se propõe a fazer;
- O compilador ignora os comentários, portanto não ocupa espaço na memória *flash*;
- Para se comentar uma única linha do código usa-se o //;
- Para se comentar um bloco de linhas, delimita-se este bloco entre /* e */;

Sketch - Exemplo



```
/*      Codigo desenvolvido para o Curso Arduino -Explorando UERJ  */

// Definição dos pinos deste sketch
const byte ledPin8=8;
const byte ledPin10=10;
const byte ledPin11=13;

void setup(){
  pinMode(ledPin10,OUTPUT);
  pinMode(ledPin8,OUTPUT);
  pinMode(ledPin11,OUTPUT);
}

void loop(){

  digitalWrite(ledPin8,HIGH);
  digitalWrite(ledPin10,LOW);
  digitalWrite(ledPin11,LOW);
  delay(500);          // Tempo de Espera 500 mseg

  digitalWrite(ledPin8,LOW);
  digitalWrite(ledPin10,HIGH);
  digitalWrite(ledPin11,LOW);
  delay(500);          // Tempo de Espera 500 mseg

  digitalWrite(ledPin10,LOW);
  digitalWrite(ledPin11,HIGH);
  digitalWrite(ledPin8,LOW);
  delay(500);          // Tempo de Espera 500 mseg
}
```

print() e println()



- Imprimem dados para uma porta serial;
- O println() só difere do print() por inserir um caracter de retorno de carro(\r), seguido por um caracter de nova linha(\n) ao final do que será impresso;
- Útil para depurar o código do *sketch*;
- Como usa muita memória deve ser usado com moderação;
- Sintaxe:
 - Serial.print(qualquer-dado-a-imprimir) ou
 - Serial.print(dado-numérico-a-imprimir,formato)

print() e println()



- Os seguintes formatos de apresentação são permitidos:
 - DEC (em decimal)
 - OCT (em octal)
 - HEX (em hexadecimal)
 - BIN (em binário)
- No caso de valores de ponto flutuante, o formato deverá indicar o número máximo de casas decimais;

Vamos praticar! Exemplo em Explorando_o_Arduino_Faperj_UERJ_print_println;

Entradas e Saídas digitais



- Por padrão, todos os pinos do Arduino são classificados como pino de INPUT;
- Se algum pino for utilizado como OUTPUT isto deve ser declarado no *sketch*;
- As entradas e saídas dos pinos digitais podem acusar, ou propagar, um sinal alto(**HIGH**) ou baixo(**LOW**);
- Os pinos de saída **PWM(~)** fazem um pouco mais;
- O Arduino tem a capacidade de detectar um sinal elétrico de apenas 10µs de duração;
- A biblioteca padrão do Arduino disponibiliza funções para manipular estes pinos;

pinMode(), digitalWrite() e digitalRead()



■ Sintaxe:

pinMode(pino,modo);

pino – identifica qual pino será utilizado;

modo – indica a finalidade do pino: INPUT, OUTPUT ou INPUT_PULLUP;

digitalRead(pino);

Retorna o valor HIGH(1) ou LOW(0);

digitalWrite(pino,valor);

valor – HIGH ou LOW;

Entradas e Saídas digitais



```
int ledPin = 13;           // LED connected to digital pin 13

void setup()
{
    pinMode(ledPin, OUTPUT); // sets the digital pin as output
}

void loop()
{
    digitalWrite(ledPin, HIGH); // sets the LED on
    delay(1000);
    digitalWrite(ledPin, LOW); // sets the LED off
    delay(1000);
}
```

Entradas e Saídas digitais



```
int ledPin = 13;           // LED connected to digital pin 13
int inPin = 7;             // pushbutton connected to digital pin 7
int val = 0;               // variable to store the read value

void setup()
{
    pinMode(ledPin, OUTPUT); // sets the digital pin 13 as output
    pinMode(inPin, INPUT);   // sets the digital pin 7 as input – desnecessário!!!
}

void loop()
{
    val = digitalRead(inPin); // read the input pin
    digitalWrite(ledPin, val); // sets the LED to the button's value
}
```

Códigos copiados de www.arduino.cc/en/Reference/DigitalRead

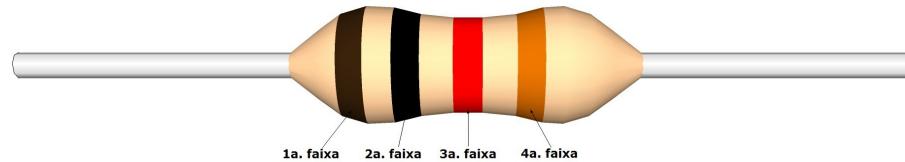
Componente Resistor



- O resistor é usado basicamente para limitar a corrente e a voltagem em partes de um circuito;
- Têm diversos valores, tamanhos e formatos;
- Dependendo do tipo, seus valores podem ser identificados por um conjunto de cores;
- Possui uma potência de dissipação nominal que é a potência que ele pode absorver em funcionamento normal;
- Símbolos do resistor:



Componente Resistor - identificação

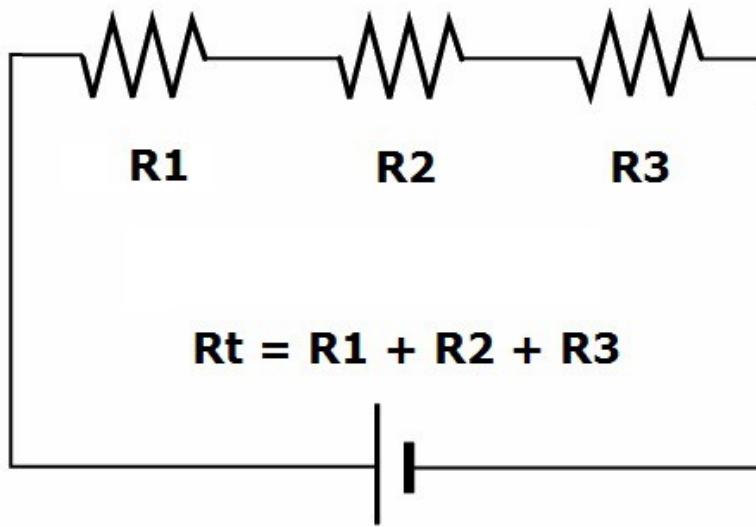


Cores	1 ^a Faixa	2 ^a Faixa	3 ^a Faixa Multiplicativa	4 ^a Faixa Tolerância	
Preto	-	0	-	-	-
Marrom	1	1	0	±1%	F
Vermelho	2	2	00	±2%	G
Laranja	3	3	000	-	-
Amarelo	4	4	0000	-	-
Verde	5	5	00000	±0,5%	D
Azul	6	6	000000	±0,25%	C
Violeta	7	7	0000000	±0,1%	B
Cinza	8	8	00000000	±0,05%	A
Branco	9	9	000000000	-	-
Ouro	-	-	÷10	5%	J
Prata	-	-	÷100	10%	K

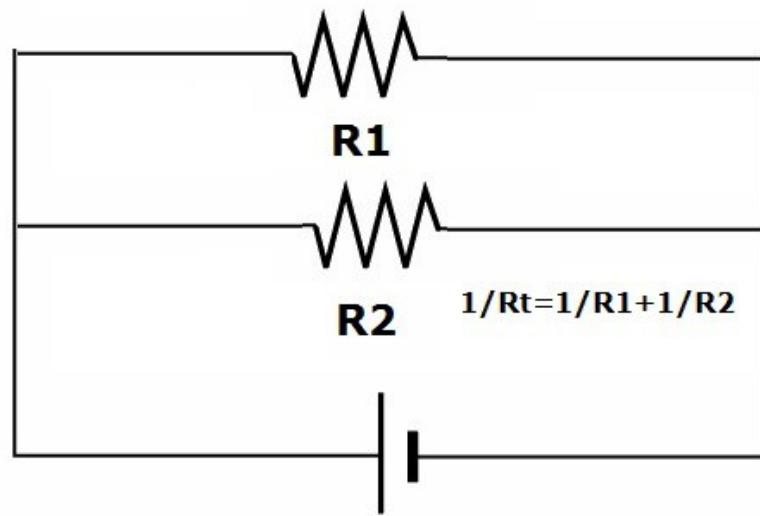
Associação de Resistores



- Utilizada quando se quer alcançar um valor de resistência não encontrado comercialmente;



Em série

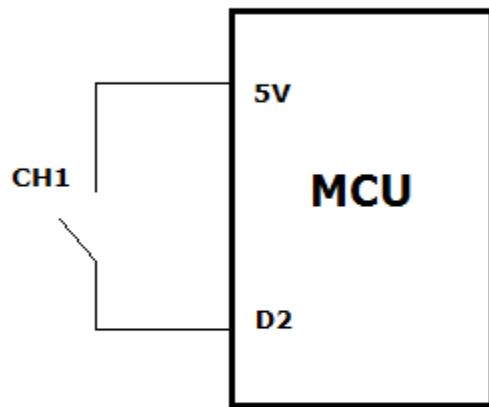


Em paralelo

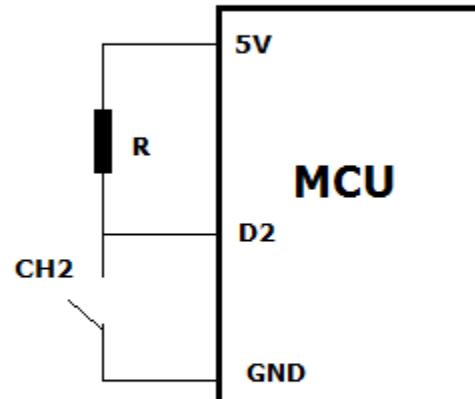
Modo INPUT_PULLUP



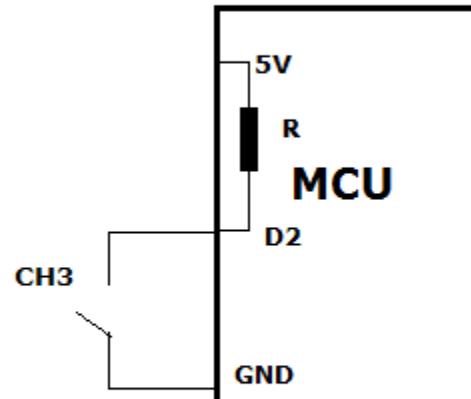
- Resistor embutido nos pinos digitais do Arduino;
- Usado para garantir o estado do sinal em 0(LOW) ou 1(HIGH);
- Quando o pinMode() indica INPUT_PULLUP o resistor é ligado a 5V da MCU;



Flutuação



Com Componentes Externos



INPUT_PULLUP

Modo INPUT_PULLUP



- Com a chave(CH3) aberta, o sinal fica HIGH, fechada o sinal fica LOW;
- No Arduino Due o valor do resistor varia de $50\text{K}\Omega$ a $150\text{K}\Omega$;
- Nos demais Arduinos, o valor varia de $20\text{K}\Omega$ a $50\text{K}\Omega$;

delay() e delayMicroseconds()



- O Arduino interrompe qualquer atividade durante o tempo especificado na função;
- Sintaxe:
 - `delay(tempo-em-milissegundos);`
 - `delayMicroseconds(tempo-em-microssegundos);`
- Utilizado para sincronizar e temporizar tarefas;
- No `delayMicroseconds()` o maior valor suportado, sem perda da acurácia, é 16383;

Operador de Atribuição



- Utilizado para atribuir um valor a uma variável;
- Representado pelo sinal de igualdade (=);
- Sintaxe:
 - nome-da-variável = expressão;
- A expressão pode ser uma constante ou outra variável;

Componente LED

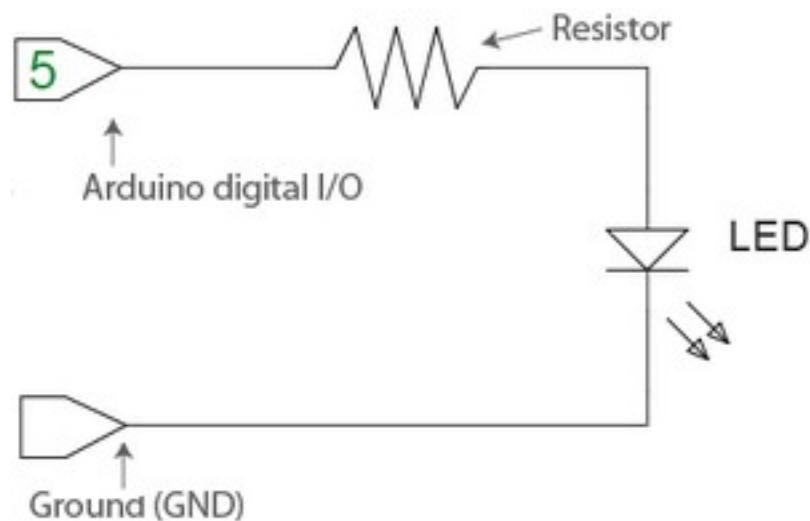
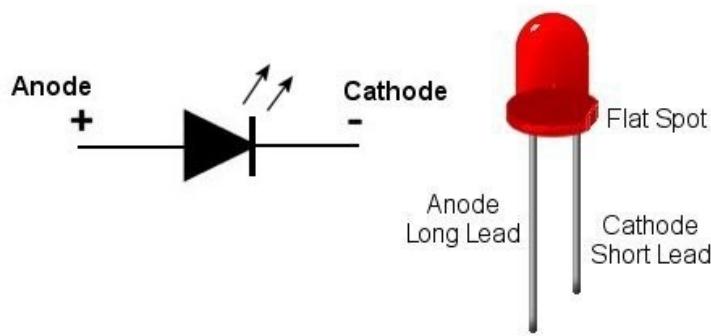


- LED é o acrônimo de Light Emitting Diode;
- O diodo tem como característica permitir a passagem da corrente elétrica somente em um única direção;
- Requer uma pequena corrente para acender;
- Quando conectado a um circuito haverá sempre uma tensão entre seus terminais;
- Valores normalmente comercializados:
 - Tensão de operação entre 1.8-2.2V;
 - Corrente de operação entre 16-18mA;
 - Corrente máxima de 20mA;

Componente LED



- Quando conectado ao Arduino, por precaução, deve ser colocado em série com um resistor;



Experimento #1

Controle de LEDs



- Projetar um sinal luminoso de trânsito com as seguintes regras;
 - A luz verde deve ficar acesa por 60s;
 - A luz amarela deve ficar acesa por 10s, concomitante aos últimos 10s da luz verde;
 - A luz vermelha deve ficar acesa por 30s;
 - O ciclo se repete indefinidamente;
- Componentes utilizados;
 - Placa Arduino UNO;
 - LEDs vermelho, amarelo e verde;
 - Jumpers e protoboard;

Vamos praticar! Exemplo em

[Explorando_o_Arduino_Faperj_UERJ_sinal_luminoso_basico](#);

Operadores aritméticos



Sintaxe	Significado
+	adição
-	subtração
*	multiplicação
/	divisão
%	resto

Operadores de composição



Sintaxe	Significado
$a+=b$	$a= a + b$
$a-=b$	$a= a-b$
$a^*=b$	$a= a^*b$
$a/=b$	$a= a/b$
$a\%=b$	$a= a\%b$
a^{++}	$a= a+1$
a^{--}	$a= a-1$

Estrutura de Controle



- Possibilita que trechos de códigos, sob determinadas condições, sejam ou não executados;
- Implementada pelas instruções if, if/else e switch case;
- Instrução if, sintaxe:
 - if (expressão-de-comparação)
 ação; ou
 - if (expressão-de-comparação) {
 ação 1;
 ação 2;
 ação n;
 }

Estrutura de Controle



■ Instrução if/else, sintaxe:

```
-      if (expressão-de-comparação) {  
          ação; ou ações;  
      }  
      else {  
          ação; ou ações;  
      } ou  
      if (expressão-de-comparação) {  
          ação; ou ações;  
      }  
      else if (expressão-de-comparação){  
          ação; ou ações;  
      }
```

Estrutura de Controle



■ Instrução switch case, sintaxe:

```
- switch (nome-da-variavel-a-ser-comparada) {  
    case x:    // verifica se o valor da variavel é igual a x  
        ação; ou ações;  
        break;  
    case y:    // verifica se o valor da variavel é igual a y  
        ação; ou ações;  
        break;  
    default:   // se verificação mal sucedida (uso opcional)  
        ação; ou ações;  
        break;  
}
```

Operadores de comparação



Sintaxe	Significado
<code>==</code>	igual a
<code>!=</code>	não igual a
<code><</code>	menor que
<code>></code>	maior que
<code><=</code>	menor ou igual que
<code>>=</code>	maior ou igual que

Operadores booleanos



- Utilizados na expressão de comparação da instrução if;
- São 3 operadores:

- And lógico:
 - &&
- Or lógico:
 - ||
- Not:
 - !

- Exemplos:

```
if (a== 10 && b>=20)  
if (a!=10)  
if (a==10 || b>=20)
```

Codificando Funções



■ Sintaxe:

```
- tipo-do-dado-retornado nome-da-função ([tipo-dado1 arg1[,tipo-dado2 arg2[,...,[tipo-dadon argn]]]]){  
    ações;  
    [return expressão; ]  
}
```

■ Caso não haja valor a ser retornado, o tipo do dado retornado deve ser declarado como *void(nulo)*;

■ Arg1, arg2,..., argn são os argumentos passados para a função;

■ Tipo-dado1,tipo-dado2,...,tipo-dadon são os respectivos tipos de dados dos argumentos;

Codificando Funções



■ Exemplo de uso:

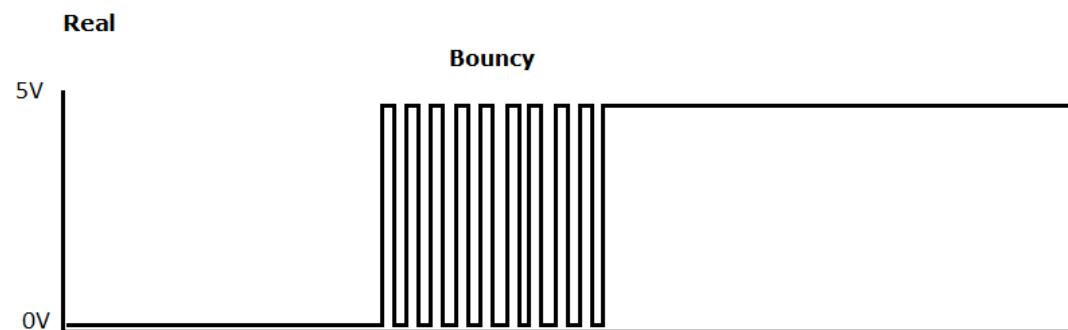
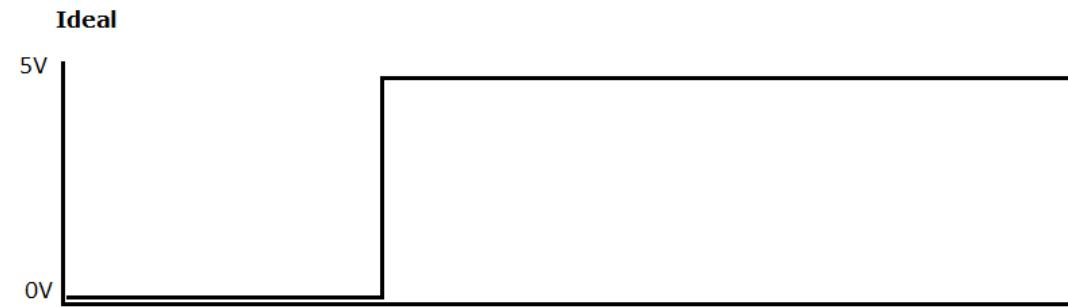
```
void loop(){
    .....
    Serial.println(converteCelsius(70.6));
}

float converteCelsius(float fahrenheit) {
    return (fahrenheit - 32)/1.8;
}
```

debouncing



- Técnica para ignorar o ruído quando uma chave mecânica é pressionada;
- Como o Arduino captura sinais com até $10\mu s$, o efeito *bouncy* desestabiliza o sinal de entrada;



debouncing



Os seguintes passos são necessários para o *debouncing*:

- 1)Armazenar o estado prévio e corrente da chave(inicia como LOW);
- 2)Lê o estado corrente da chave;
- 3)Se o estado corrente difere do estado prévio, espera 5ms;
- 4)Após 5ms lê novamente o estado tornando-o o estado corrente;
- 5)Se o estado prévio era LOW e o estado corrente é HIGH, alterna o valor da saída;
- 6)Troca o valor do estado prévio para o valor do estado corrente;
- 7)Retorna para o passo 2;

debouncing

Exemplo



```
// Código extraído, e adaptado, do livro Exploring Arduino – Jeremy Blum

const int botao=2;
const int led=9;

boolean estadoCorrente = LOW;
boolean estadoAnterior=LOW;
boolean ledOn = false;

void setup() {
    pinMode(led,OUTPUT);
    // Não utilizou o recurso de INPUT_PULLUP , logo tem que usar um resistor externo(10K).
}

void loop() {

    estadoCorrente = debounce(estadoAnterior);
    if (estadoAnterior==LOW && estadoCorrente==HIGH){
        ledOn = !ledOn;
    }
    estadoAnterior = estadoCorrente;
    digitalWrite(led,ledOn);
    }

boolean debounce(boolean estadoAnterior){          // Função debounce

    boolean corrente = digitalRead(botao);
    if( estadoAnterior!= corrente){
        delay(5);
        corrente = digitalRead(botao);
    }
    return corrente;
}
```

Experimento #2

Controle de LEDs



- Implementar um sinal luminoso, com botão, para travessia de pedestres.
Regras:

- A luz verde fica normalmente acesa;
- Quando o botão é pressionado, 20 seg depois a luz amarela acende, concomitante à luz verde. Ficarão acesas durante 10s;
- A luz vermelha acende, permanecendo assim por 20s;
- A luz verde volta a acender;
- A luz amarela só acende se a luz verde já estiver acesa;

- Componentes utilizados;

- Placa Arduino UNO;
- Uma micro-chave mecânica;
- LEDs vermelho, amarelo e verde;
- Jumpers e protoboard;

Obs: utilizar o resistor INPUT_PULLUP e **debouncing**.

Vamos praticar! Exemplo em Explorando_o_Arduino_Faperj_UERJ_sinal_luminoso_chave;

Entradas Analógicas

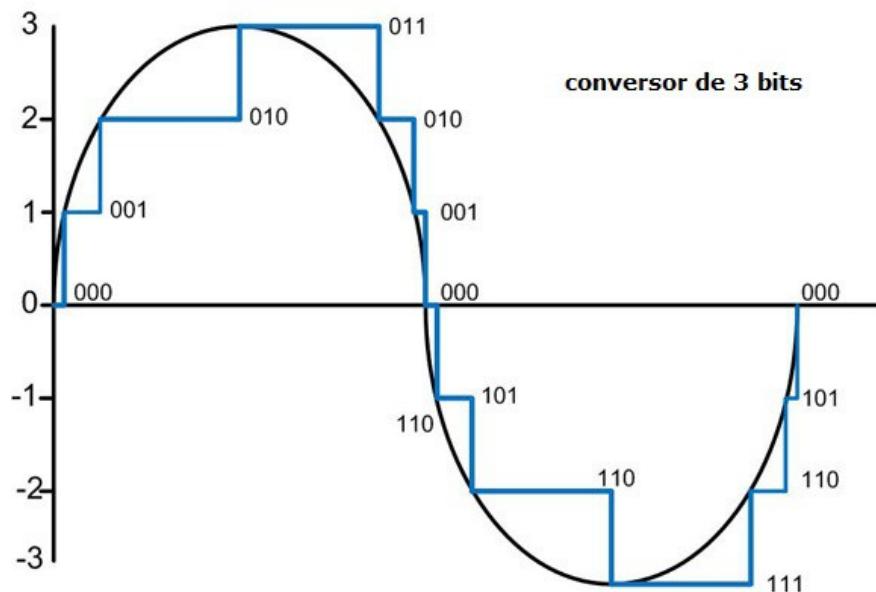


- Sinais analógicos são aqueles que não podem ser discretamente classificados;
- Podem ter um infinito número de valores, em um único intervalo;
- Um computador não pode medir um número infinito de casas decimais, possíveis em um sinal analógico;
- O Arduino converte o sinal analógico para uma representação digital(*analog-to-digital-converter*);
- Usa uma resolução de 10 bits($2^{10} = 1024$) para esta operação;

Entradas Analógicas



- O Arduino atribui um valor entre 0 e 1023 para qualquer valor analógico lido;
- Um valor de entrada de 5V(voltagem de referência) corresponderia a 1023, de 0V a 0, de 2.5V a 512;



PWM



- PWM é o acrônimo de *Pulse Width Modulation*;
- Utilizado para produzir sinais que emulam saídas analógicas;
- Os pinos aptos à PWM são identificados pelo sinal ~;
- Na família UNO são os pinos 3, 5, 6, 9, 10 e 11;
- A saída PWM usa 8bits($2^8 = 256$) para esta operação;
- A saída, então, produzirá um sinal com largura de pulso entre 0 e 255;
- 255 corresponde a 5V, 122 a 2.5V, 0 a 0V;

PWM

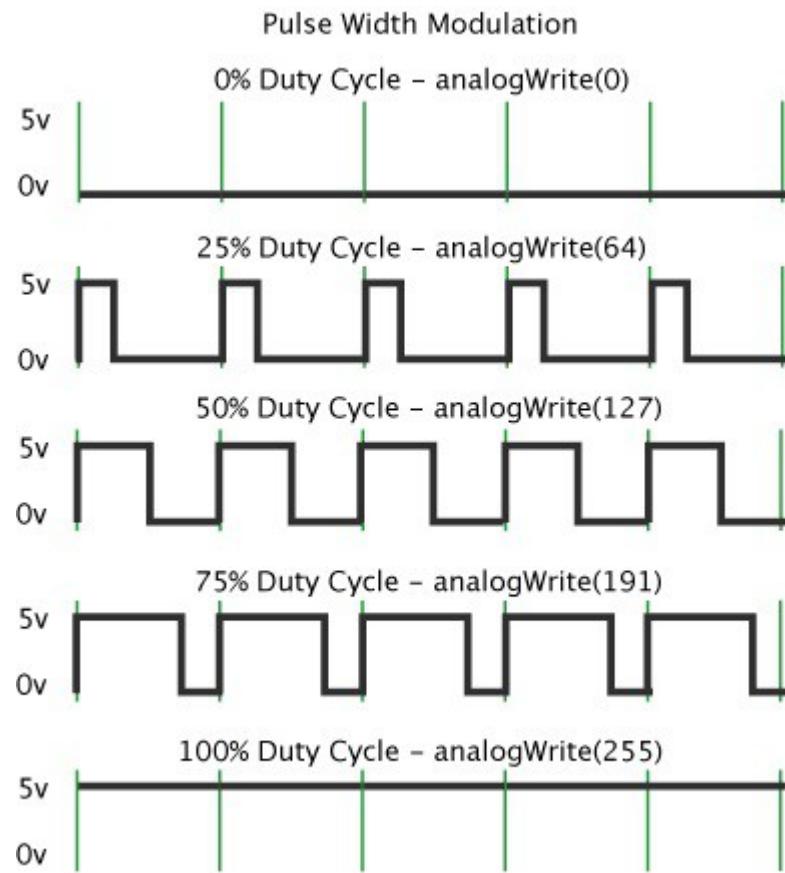


Figura extraída de <https://www.arduino.cc/en/Tutorial/PWM>

analogWrite() e analogRead()



■ Sintaxe:

analogWrite(pino,valor);

pino – produz um sinal PWM neste pino;

valor – valor da largura de pulso, entre 0 e 255;

analogRead(pino);

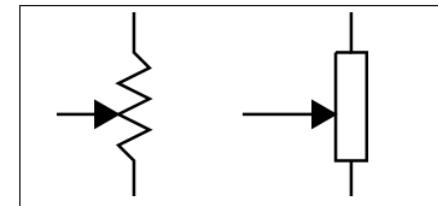
retorna um valor entre 0 e 1023;

Obs.: a frequência do sinal PWM em seus pinos é de 490hz, com exceção dos pinos 5 e 6 que é 908hz;

Componente Potenciômetro



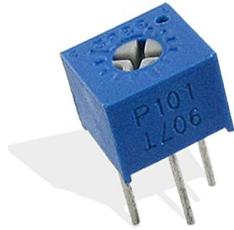
- É um resistor com valor variável de resistência;
- Apresentado em diferentes formas, tamanhos e valores;
- Possui 3 terminais;



Componente Potenciômetro



- Potenciômetro e trimpots;
- Uma extremidade liga-se ao VCC, a outra à GND;
- Filamento central é o sinal!



map() e constrain()



- Função map();
- Utilizada para mapear faixas de valores em outras faixas;
- Sintaxe:

```
map (nome-da-variavel,menor-valor-faixa-origem,maior-valor-faixa-  
origem,menor-valor-faixa-alvo,maior-valor-faixa-alvo);
```

- Exemplo:

```
val = analogRead(5);  
  
val = map (val,0,1023,0,255);  
  
analogWrite(6,val);
```

map() e constrain()



- Função constrain();
- Utilizada para restringir um número em uma faixa;
- Sintaxe:
 - constrain (nome-variavel,menor-valor-faixa,maior-valor-faixa);
- Exemplo:

```
val = analogRead(5);
      val = constrain (val,10,500);
```

- Caso $\text{val} < 10$, o valor de val será ajustado para 10;
- Caso $\text{val} > 500$ o valor de val será ajustado para 500;

Experimento #3

Sinais analógicos



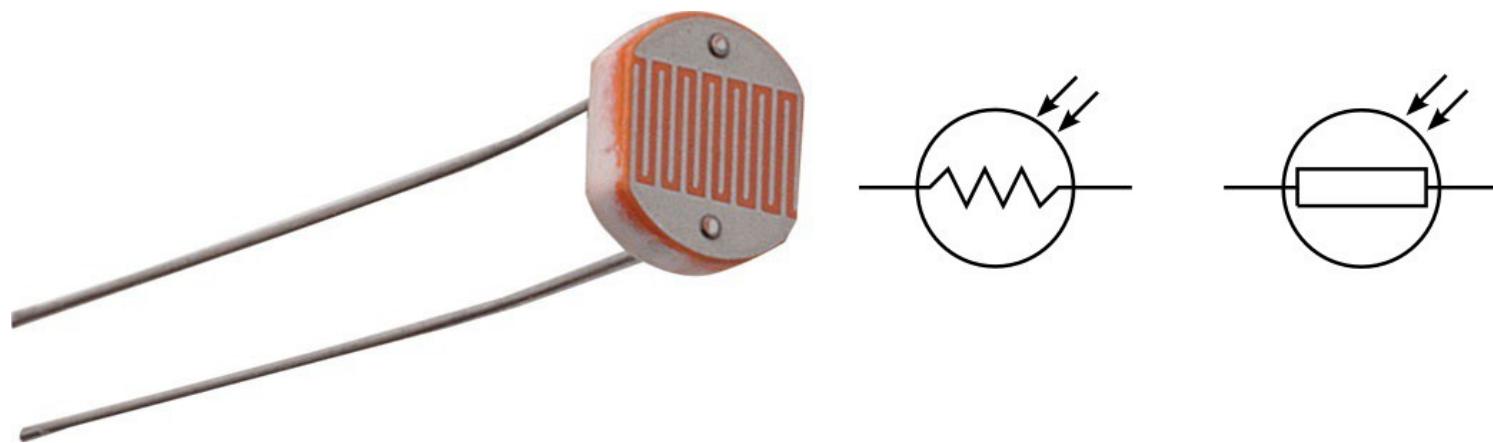
- Controlar a luminosidade de um LED utilizando um resistor variável;
- Componentes utilizados;
 - Placa Arduino UNO;
 - 1 LED;
 - 1 trimpot de 10k;
 - Jumpers e protoboard;

Vamos praticar! Exemplo em [Explorando_o_Arduino_Faperj_UERJ_potenciometro](#);

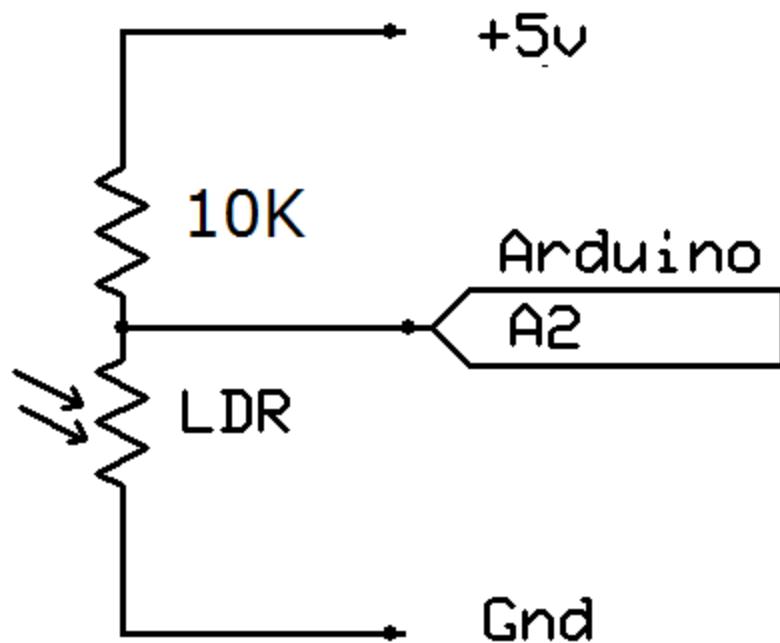
Componente LDR



- LDR é o acrônimo de Light Dependent Resistor;
- Sua resistência é inversamente proporcional a quantidade de luz percebida;
- Em geral, pode variar de 400Ω até $1M\Omega$;
- Age como um sensor analógico de luminosidade;



Componente LDR



Circuito sensor de luminosidade

Experimento #4

Ligar-desligar um LED com um LDR



■ Ligar ou desligar um LED de acordo com a luminosidade do ambiente, segundo as regras:

- Se o valor indicado pelo LDR for maior do que 850, o LED acenderá(**atenção à luminosidade do LED!!!**);
- Para valores menores, o LED ficará apagado;

■ Componentes utilizados;

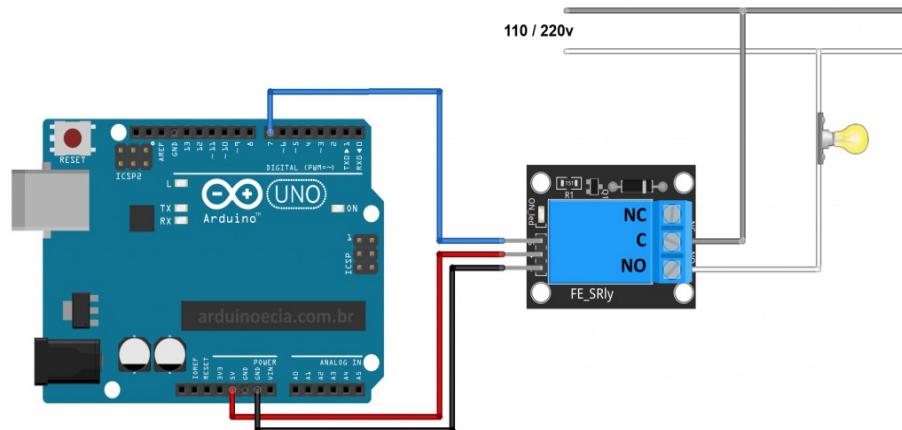
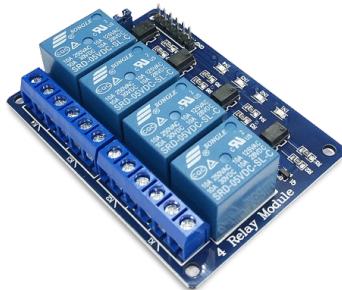
- Placa Arduino UNO;
- 1 LED;
- 1 LDR;
- 1 resistor de 10k;
- Jumpers e protoboard;

Vamos praticar! Exemplo em Explorando_o_Arduino_Faperj_UERJ_LDR_LED;

Componente Relé



- Dispositivo utilizado para fechar/abrir qualquer circuito elétrico;
- Sensibilizado por sinal um HIGH ou LOW;
- Pode ser eletromecânico ou de estado sólido;
- Ficar atento às especificações do dispositivo: Tensão e corrente máximas;



NC = Normalmente Fechado; **NO** = Normalmente Aberto; **C** = Comum;

Experimento #4.1

Abrir-Fechar Relé com um LDR



■ Abrir ou fechar um Relé de acordo com a luminosidade do ambiente, segundo as regras:

- Se o valor indicado pelo LDR for maior do que 850, o Relé fechará;
- Para valores menores, o Relé ficará aberto;

■ Componentes utilizados;

- Placa Arduino UNO;
- 1 Relé;
- 1 LDR;
- 1 resistor de 10k;
- Jumpers e protoboard;

Vamos praticar! Sem exemplo;

Estrutura de Repetição



- Possibilita a execução continuada de trechos de códigos;
 - Implementada pelas instruções for, while e do while;
 - Instrução for, sintaxe:
 - `for (tipo valor-inicial; condição-de-repetição; incremento-ou-decremento) {
 ação; ou ações;
}`
- Onde :
- valor-inicial - especifica o valor de partida para o número de repetições do conjunto de ações;
 - condição-de-repetição - indica em qual condição a execução será executada;
 - incremento-ou-decremento - indica o valor da parcela que será incrementada, ou decrementada, do valor inicial;

Estrutura de Repetição



■ Exemplos;

- _

```
for (int i=0; i<=200; i++) {  
    Serial.println(i);  
}  
_ for (int i=200; i>=0; i--){  
    Serial.println(i);  
}  
_ for (int i=10; i<50; i+=2){  
    Serial.println(i);  
}
```

Estrutura de Repetição



■ Instrução while, sintaxe:

- `while (condição-de-repetição) {
 ação; ou ações;
}`

Onde :

- condição-de-repetição – indica em qual condição a execução será executada;

■ Instrução do while, sintaxe:

- `do {
 ação; ou ações;
} while(condição-de-repetição);`

Estrutura de Repetição



■ Exemplos;

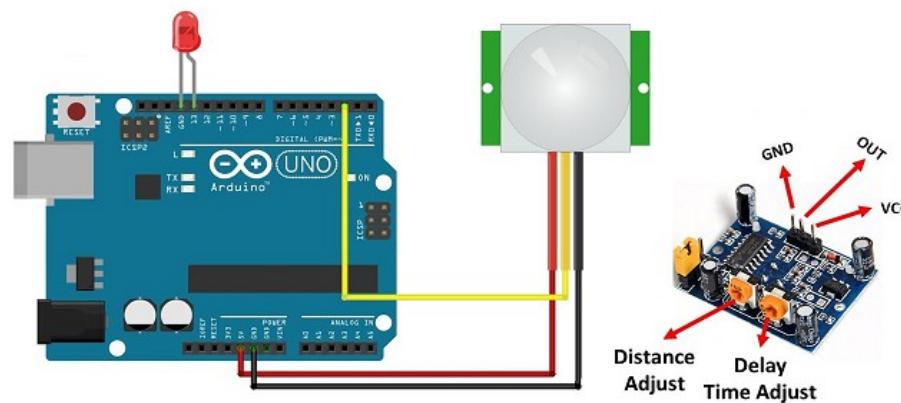
- ```
while (i>200) {
 Serial.println(i);
 i++;
}
– do{
 Serial.println(i);
 i++;
} while (i>200);
```

**Vamos praticar!** Alterar Explorando\_o\_Arduino\_Faperj\_UERJ\_print.println;

# Sensor de Movimento HC-SR501



- Módulo que usa um sensor PIR(piroelétrico), capaz de detectar a variação de calor emitida por algum animal de “sangue quente”;
- Mais calor implica em mais emissão de raios infravermelhos;
- Potenciômetros para ajustes de tempo(5s a 2,5 min) e distância(3 a 7m);
- Seu ângulo de abertura é de aproximadamente 120º;
- Alimentação: de 5V a 20V, tensão de Saída: 3,3V;



# Experimento #5

## Ligar um LED por sensor PIR



- Ligar um LED quando for detectado algum movimento detectado pelo sensor PIR;
- O LED ficará ligado durante 15s, depois se apaga;
- Componentes utilizados;
  - Placa Arduino UNO
  - 1 LED;
  - 1 Sensor PIR;
  - Jumpers e protoboard;

**Vamos praticar!** Exemplo em Explorando\_o\_Arduino\_Faperj\_UERJ\_sensor\_presenca\_LED;

# Experimento #5.1

## Abrir-fechar um relé por sensor PIR



- Abrir ou fechar um relé quando for detectado algum movimento detectado pelo sensor PIR;
- O relé ficará fechado durante 15s, depois abre;
- Componentes utilizados;
  - Placa Arduino UNO
  - 1 Relé;
  - 1 Sensor PIR;
  - Jumpers e protoboard;

**Vamos praticar!** Sem exemplo;

# Bibliotecas



- Bibliotecas são um conjunto de funções pré-codificadas com um propósito final específico;
- Facilita a tarefa do programador poupando tempo de pesquisa e codificação;
- A tarefa já vem pronta;
- Invocadas através da diretiva `#include`, inserida na área global do *sketch*;
- As bibliotecas do Arduino/core são automaticamente incluídas no *sketch*;
- A biblioteca Wiring, inspiradora do Arduino, foi criada para dar suporte programático no uso da pinagem da MCU Atmel;

# Bibliotecas padrão



|                 |                                                                                                                  |
|-----------------|------------------------------------------------------------------------------------------------------------------|
| EEPROM          | Para armazenamento de dados na EEPROM                                                                            |
| Ethernet        | Para comunicação usando o protocolo TCP/IP                                                                       |
| Firmata         | Envia comandos seriais para computador, para tratar níveis de sinais nos pinos digitais e ler valores analógicos |
| LiquidCrystal   | Para display LCD alfanuméricicos                                                                                 |
| SD              | Para ler e gravar dados em cartões SD                                                                            |
| Servo           | Para controlar servomotores                                                                                      |
| SPI             | Para usar a Serial Peripheral Interface                                                                          |
| Software Serial | Permite usar quaisquer dois pinos, não seriais, para enviar/receber dados                                        |
| Stepper         | Para controlar motores de passo                                                                                  |
| WiFi            | Para acesso e uso da rede WiFi                                                                                   |
| Wire            | Para comunicação I2C com os periféricos                                                                          |

# Bibliotecas de terceiros



|                            |                                                                             |
|----------------------------|-----------------------------------------------------------------------------|
| OneWire                    | Para leitura de dados de dispositivos que usam apenas 1-fio                 |
| RTC                        | Para comunicação com dispositivos do tipo real-time clock                   |
| Dallas Temperature Control | Para comunicação com o sensor de temperatura DS18B20                        |
| Virtual Wire               | Fornece comunicação serial entre dois Arduinos com uma frequencia de 433Mhz |
| IRRemote                   | Envia e recebe comandos remotos IR                                          |

# Motores DC



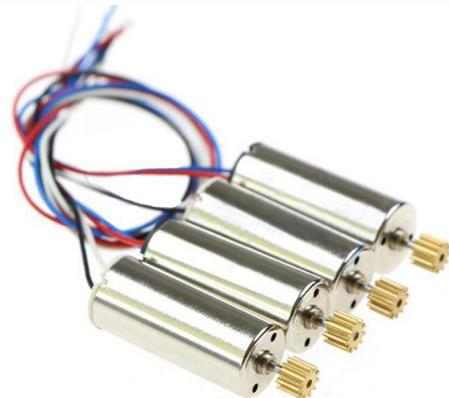
- São aqueles que funcionam alimentados por uma fonte de corrente contínua;
- Apresentado em diferentes formas, tamanhos e potências;
- Importante saber:
  - Velocidade de rotação(RPM);
  - Torque (força que o motor consegue gerar);
  - Faixas de voltagem e corrente operacionais;
- Diminuindo a voltagem diminuirá a rotação;
- Mudando a polaridade da voltagem, muda o sentido de rotação;

# Motores DC



- Para aumentar o torque pode ser usado uma caixa de engrenagens(Motorreductor);
- A saída PWM é usada para controlar a RPM;
- Problema: a potência entregue pelo Arduino pode ser insuficiente;
- Solução: uso de transistores e fontes alternativas;
- Os motores são categorizados em com ou sem escovas;
- Os motores DC simples são usados em:
  - Bombas peristálticas(equipamentos hospitalares)
  - Bombas centrífugas (aquários e lagos)
  - Veículos rádio controlados
  - Automóveis elétricos(Tesla Motors)

# Motores DC



# ServoMotores DC



- Em um único invólucro encontram-se:
  - Um motor DC;
  - Engrenagens;
  - Circuito de controle;
- Por comando, podem girar para determinada posição angular;
- Podem permanecer estáticos, nesta posição;
- Em geral, giram em torno de seu eixo de 0° a 180°;
- A maioria trabalha com alimentação de 5V;
- São chamados de servos de *hobby*;

# ServoMotores DC

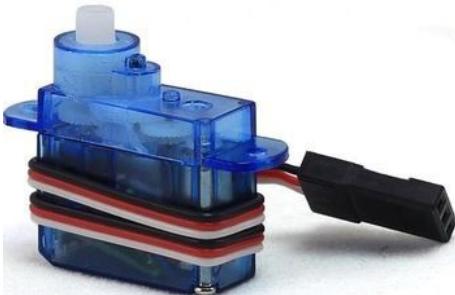


- É possível saber a posição angular;
- Utilizado em projetos de precisão;
- Possuem três terminais:
  - Voltagem(em geral, 5V);
  - Terra;
  - Controle(em geral, 5V. **PWM**);
- São controlados usando a técnica de largura de pulsos:
  - Sinal de 1ms 5V → 0°
  - Sinal de 1.25ms 5V → 45°
  - Sinal de 1,5ms 5V → 90°
  - Sinal de 2ms 5V → 180°

# ServoMotores DC



- Para a posição ficar estável é necessário reenviar o sinal a cada 20ms;
- Para facilitar, existe a biblioteca padrão Servo;



# Experimento #6

## Movimentar um servo-motor



- Movimentar um servo-motor como um limpador de pára-brisas;
- Utilização da biblioteca Servo.h(**saída PWM**);
- Componentes utilizados;
  - Placa Arduino UNO
  - 1 Servo-motor;
  - Jumpers;

**Vamos praticar!** Exemplo em Explorando\_o\_Arduino\_Faperj\_UERJ\_Servo\_Motor;

# Experimento #6.1

## Movimentar um servo-motor



- Controlar a posição de um servo-motor usando um trimpot(pode fazer em dupla!);
- Ler o valor do potenciômetro(0-1023) e mapear para o servo(0-180);
- Antes de movimentar, esperar estabilizar a posição;
- Componentes utilizados
  - Placa Arduino UNO
  - 1 Trimpot;
  - 1 Servo-motor;
  - Jumpers e protoboard;

**Vamos praticar!** Sem Exemplo;

# Vibração DC



- São micromotores que vibram quando submetidos a uma voltagem;
- Utilizado em:
  - Smartphones;
  - Brinquedos;
  - Engenhocas;
- Tensão de operação entre 2,5 e 4V;
- Corrente máxima de 90mA;



# StepperMotor DC



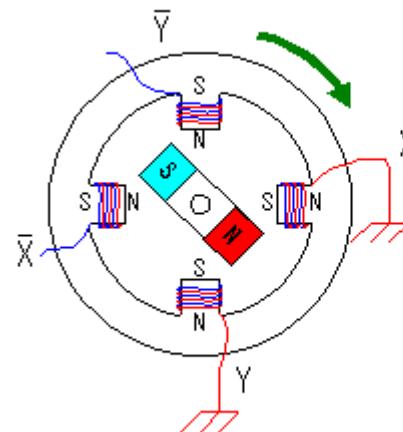
- Não usa escovas;
- Divide uma rotação completa em número de etapas;
- Em geral, energizado por um circuito externo;
- Não precisa de mecanismo para informar a posição corrente;
- Podem girar 360° em ambas as direções;
- Algumas vantagens:
  - Precisão de posicionamento;
  - Move-se para frente ou para trás, em etapas;
  - Alto torque e baixa rotação;
  - Rápida resposta às ordens de movimento angular;

# StepperMotor DC



- São utilizados em:
  - Impressoras;
  - *Disk drives*(posicionamento de cabeças);
- Podem ser unipolares ou bipolares;
- Típicas voltagens de operação: 5V, 9V, 12V e 24V;
- Para facilitar, existe a biblioteca padrão Stepper;

# StepperMotor DC

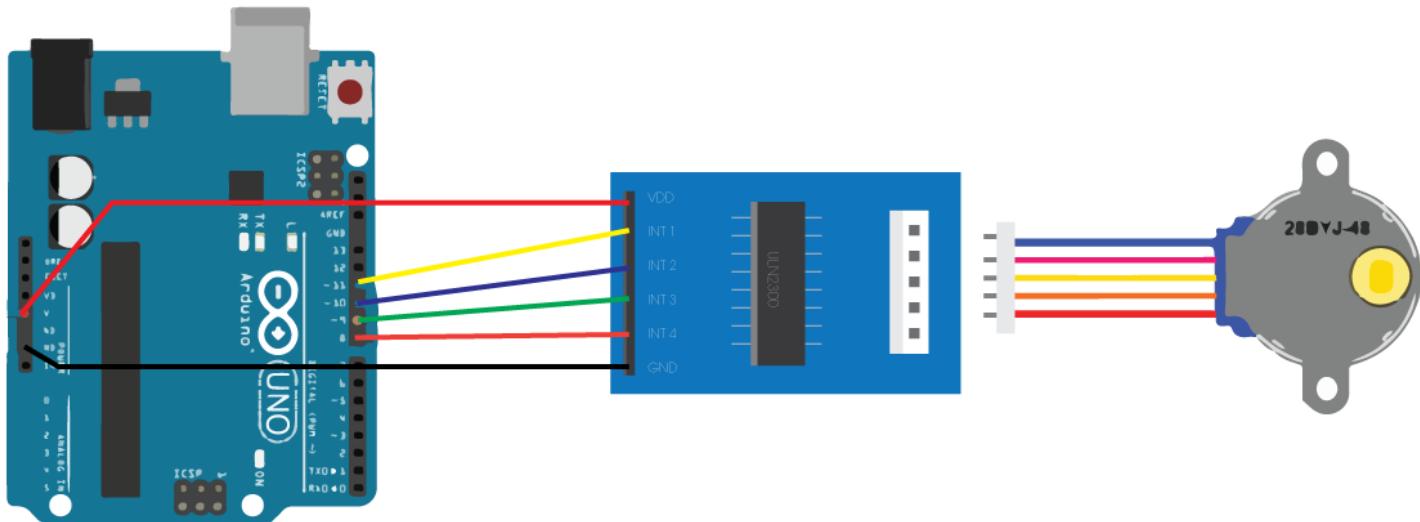


| X | $\bar{X}$ | Y | $\bar{Y}$ |
|---|-----------|---|-----------|
| 0 | 1         | 0 | 1         |
| 1 | 0         | 0 | 1         |
| 1 | 0         | 1 | 0         |
| 0 | 1         | 1 | 0         |

# Exemplo



Controlando um stepper motor.



# Exemplo



```
#include <Stepper.h>

const int etapasPorRotacao = 768;

//Inicia a biblioteca utilizando as portas de 8 a 11
Stepper stepper(etapasPorRotacao, 8,10,9,11);

void setup() {
 stepper.setSpeed(20); //Determina a velocidade inicial do motor
}

void loop() {
 // Comanda uma volta completa
 stepper.step(-2048);
 delay(2000);

 // Comanda uma volta completa em sentido contrario
 stepper.step(2048);
 delay(2000);

 // Comanda uma volta completa em 4 passos de 90 graus
 for (int i=0; i<4; i++){
 stepper.step(-512);
 delay(2000);
 }

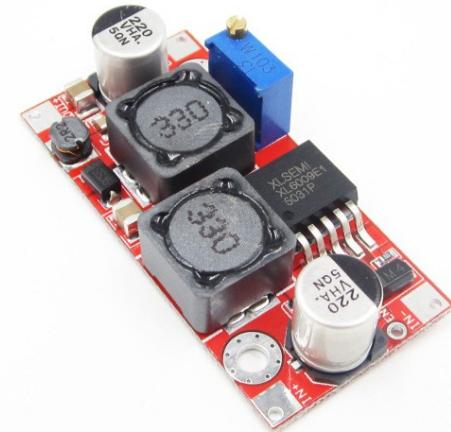
 // Comanda uma volta completa em 48 passos de 45 graus
 for (int i=0; i<8; i++){
 stepper.step(256);
 delay(2000);
 }
}
```

# Motores DC - Potência



- Muitos destes motores consomem uma potência que o Arduino não pode entregar;
- Opção é usar fontes de alimentação alternativas;
- Existem inúmeras fontes no mercado;
- Umas de voltagens reguláveis, outras de valor fixo;
- Pode-se conjugar uma fonte fixa(baixo custo) com um circuito regulável *step-down* ou *step-up*(baixo custo);
- Motores criam picos de voltagem que podem prejudicar o Arduino;

# Motores DC Fontes e circuito *step-down*



# Transistores

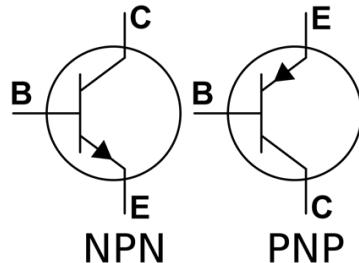
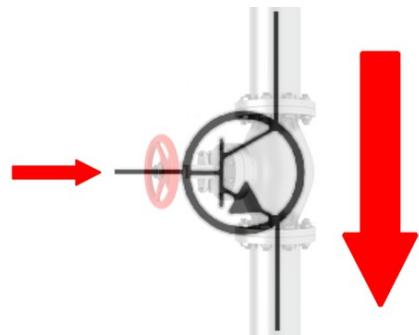


- Componente eletrônico que pode agir como:
  - Um amplificador de sinal;
  - Chave eletricamente controlada;
- Utilizado para isolar um motor DC do Arduino;
- Possui três terminais:
  - base(B);
  - coletor(C);
  - emissor(E);
- Uma pequena corrente em B pode chavear o fluxo de corrente entre C e E;

# Transistores



- Possui várias formas e tamanhos;
- Feitos, na maioria, de silício;
- Classificados por seu tipo de impureza em NPN ou PNP;
- O tipo determina como deve ser ligado a um circuito;



# Transistores

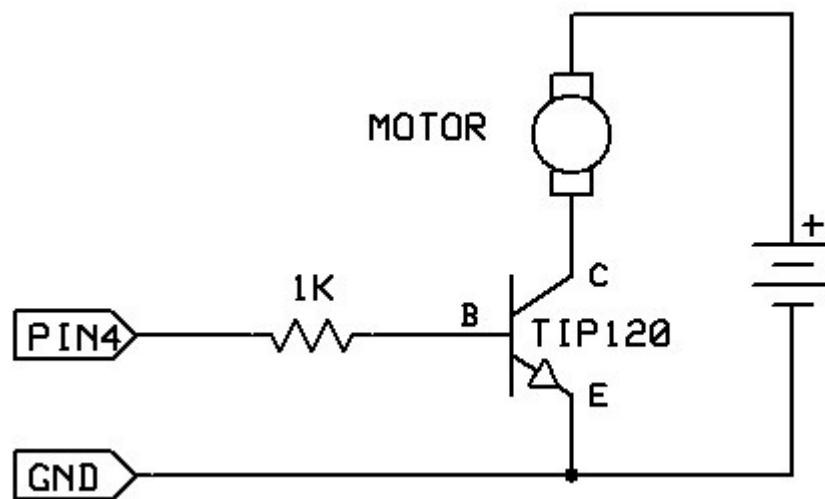


- No NPN a base(B) precisa ser conectada a uma voltagem positiva;
- No PNP a base(B) precisa ser conectada à terra;
- O transistor permitirá ao Arduino chavear o motor seguramente;
- O transistor controlará a velocidade do motor comandada pela saída PWM;

# Transistores



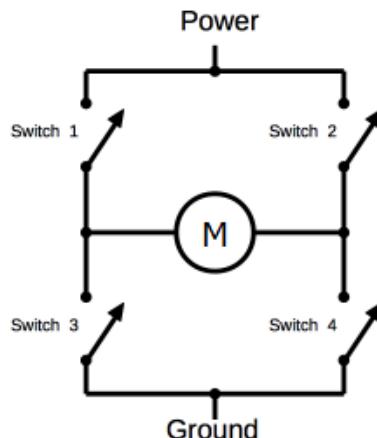
- Quando aplicada uma voltagem em B, pelo PIN4, uma corrente flui de C para E;
- Os 5V do Arduino são suficientes para gerar corrente em B;
- A saída PWM(PIN4) provoca o ligar-desligar do transistor;
- O terra(GND) do Arduino e da fonte alternativa devem ser compartilhados;



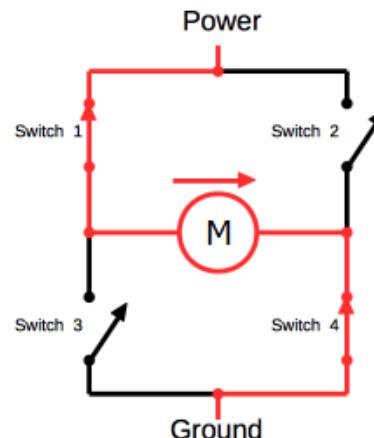
# H-Bridge



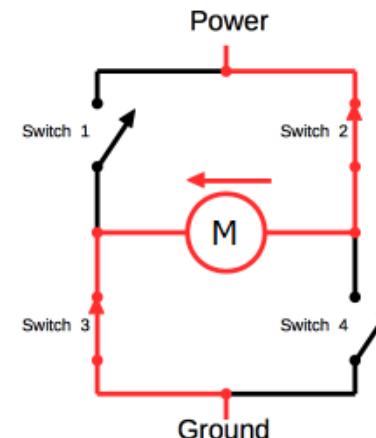
- Utilizado em motores DC simples;
- Permite girar o motor DC em ambas as direções;
- Utiliza o recurso de chaves combinadas;
- Pode ser implementada pelo *chip* SN754410;
- Este *chip* pode controlar até 2 motores;



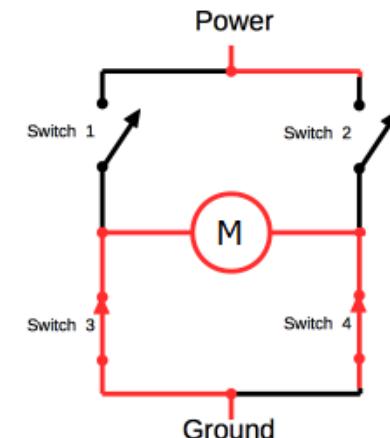
Aberto



Pra Frente



Pra trás



Parado

# Chip SN754410



GND(Pinos 4,5,12,13) – conectar à terra comum

VCC2(Pino 8) – conectar a uma outra fonte, caso a potência do Arduino sejam insuficiente

VCC1(Pino 16) - conectar ao Arduino

1Y e 2Y(Pinos 3 e 6) – conectar aos terminais do motor da esquerda

1A e 2A(Pinos 2 e 7) – chaves do motor da esquerda, conectar ao Arduino

1,2EN(Pin 1) – usado para controlar a velocidade do motor da esquerda, conectar ao PWM Arduino

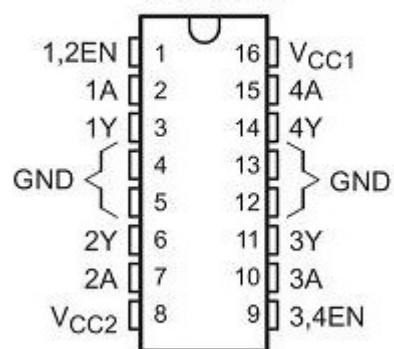
3Y e 4Y(Pinos 11 e 14) - conectar aos terminais do motor da direita

3A e 4A(Pinos 10 e 15) – chaves do motor da direita, conectar ao Arduino

3,4EN(Pino 9) - usado para controlar a velocidade do motor da direita, conectar ao PWM Arduino

## SN754410

(TOP VIEW)



# Exemplo



- Regulando a velocidade e direção de apenas um motor DC com um potenciômetro e o chip SN754410;
- Para facilitar o entendimento do *sketch* serão criadas as funções praFrente(), praTras() e parar();
- Valores do potenciômetro: entre 0-461, pra trás; entre 562-1023 pra frente; entre 462 e 561, parar;

```
void praFrente (int quanto){
 digitalWrite(EN, LOW);
 digitalWrite(MC1, HIGH);
 digitalWrite(MC2, LOW);
 analogWrite(EN, quanto);
}
void praTras (int quanto){
 digitalWrite(EN, LOW);
 digitalWrite(MC1, LOW);
 digitalWrite(MC2, HIGH);
 analogWrite(EN, quanto);
}
void parar (){
 digitalWrite(EN, LOW);
 digitalWrite(MC1, LOW);
 digitalWrite(MC2, LOW);
 digitalWrite(EN, HIGH);
}
```

# Exemplo



```
// Código extraído do livro Arduino Exploring – Jeremy Blum

const int EN=9; //H-Bridge 1 habilitar/desabilitar
const int MC1=3; //Motor Control 1
const int MC2=2; //Motor Control 2
const int POT=0; //POT no pino A0

int val = 0; //valor lido do POT
int quanto = 0; //valor da velocidade (de 0-255)

void setup(){
 pinMode(EN, OUTPUT);
 pinMode(MC1, OUTPUT);
 pinMode(MC2, OUTPUT);
 parar(); //inicia com o motor parado
}
void loop(){
 val = analogRead(POT);
 if (val >= 562) {
 quanto = map(val, 562, 1023, 0, 255);
 praFrente(quanto);
 }
 else if (val < 462) {
 quanto = map(val, 461, 0, 0, 255);
 praTras(quanto);
 }
 else {
 parar();
 }
}
```

# Exemplo



```
void praFrente (int quanto){
 digitalWrite(EN, LOW);
 digitalWrite(MC1, HIGH);
 digitalWrite(MC2, LOW);
 analogWrite(EN, quanto);
}
void praTras (int quanto){
 digitalWrite(EN, LOW);
 digitalWrite(MC1, LOW);
 digitalWrite(MC2, HIGH);
 analogWrite(EN, quanto);
}
void parar (){
 digitalWrite(EN, LOW);
 digitalWrite(MC1, LOW);
 digitalWrite(MC2, LOW);
 digitalWrite(EN, HIGH);
}
```

// Para evitar danos ao motor  
// Para evitar danos ao motor  
// Para evitar danos ao motor