INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO ACRE

EZEQUIEL SOARES DA SILVA

ARDUINO

Rio Branco

2021

EZEQUIEL SOARES DA SILVA

ARDUINO

Trabalho em caráter avaliativo para composição de nota na disciplina de Linguagens de Programação no Curso Tecnólogo em Sistemas para Internet do Instituto Federal de Educação, Ciência e Tecnologia do Acre, Campus Rio Branco.

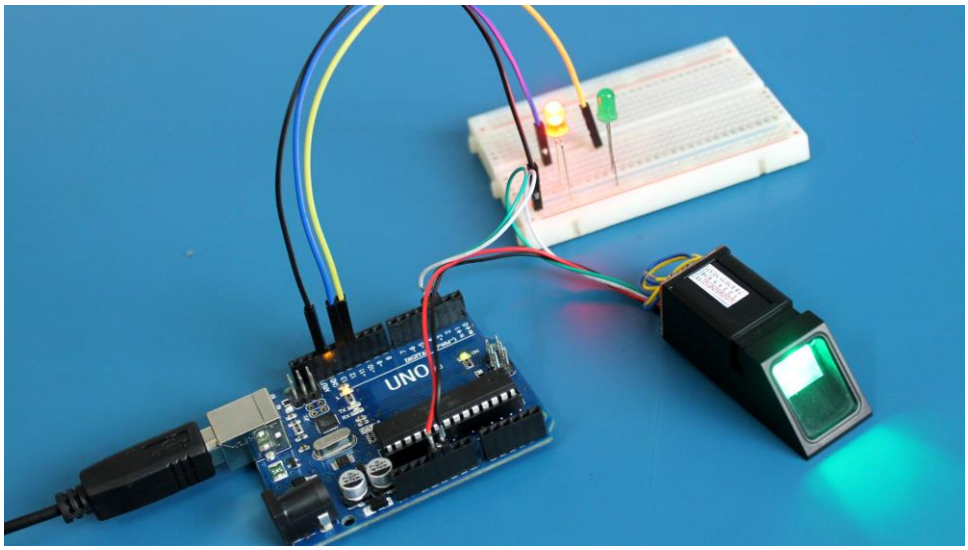
Rio Branco

2021

SUMÁRIO

1 INTRODUÇÃO

Nesta apostila será abordado o Arduino, que seria uma plataforma, tanto eletrônica, quanto programática para fazer aplicações de diferentes tipos. Ela ficou popular por causa disso, muitos programadores e design se aventura em fazer aplicações com Arduino pela sua acessibilidade e facilidade. Veja um exemplo de projeto:



1.1 Arduino biometria exemplo - usinainfo

Você pode ver na imagem 1.1 um leitor de biometria, tudo isso é controlado pelo microcontrolador do Arduino, e a parte elétrica dos componentes é conectada a ele para ser controlado. Essa facilidade fez o Arduino quem ele é hoje. A sua programação é bem parecida a linguagens famosas no mercado, e de fácil aprendizado. Nesta apostila será abordado as suas portas eletrônicas e sua programação, retirado da fonte oficial do Arduino.

2 DESENVOLVIMENTO

Se você não tiver um modelo de Arduino não se preocupe, no tópico 2.5.2 terá passos para fazer simulações usando Arduino totalmente online, mas se você tiver ele em mãos também irá conter como desenvolver e compilar direto no modelo físico. Agora veremos o que é a tecnologia, como funciona e seus paradigmas.

2.1 HISTÓRICO

O Arduino teve sua primeira versão em 2005, por meio da "*Interactive Design Institute*", em Ivrea - Itália. Seus dois idealizadores Massimo Banze e David Cuartielle habiam percebido que os produtos existentes no mercado eram caros e relativamente difíceis de usar. Então eles decidiram desenvolver um microcontrolador que poderia ser utilizado pelos seus estudantes de arte e design em seus projetos. Então o projeto foi desenvolvido, e dado no nome Arduino, em referência a um bar local frequentado por membros do corpo docente e alunos do instituto.

Depois disso as placas eram vendidas em forma de kits, para estudando exercer seus projetos de maneira flexível, fácil de manusear e barato, onde pessoas interessadas em criar sistemas e ambientes interativos sem requirir muito esforços. Por essas vantagens o produto foi se popularizando entre outros programadores e designers, assim aumentando a demanda e a produção teve que ser estendida, e hoje o Arduino é referência para prototipação de sistemas eletrônicos programáticos.

Mas afinal, o que é o Arduino? Ele torna possível o manuseio de eletrônica e programação usando a sua placa, onde há um microcontrolador de baixo custo que possibilita tudo isso, circuitos de entrada/saída que pode ser conectada à um computador e ser programado em uma IDE. A vantagem do Arduino é ele ser barato, ótima curva de aprendizado, modular (várias outras placas onde você pode expandir), etc. Nesta apostila irá considerar o Arduino Uno R3, mas existe milhares de outras versões para cada exigência.

Além disso o Arduino é *open-source* e *hardware livre*. Isso quer dizer que todo mundo tem acesso, tanto no seu código e também no seu hardware, podendo replicar e até vender. Então várias empresas começaram a produzir e vender suas próprias versões do Arduino, assim fomentando a concorrência e ficando mais barato para o consumidor final. A sua acessibilidade fez com que muitas pessoas tivessem acesso a essa tecnologia, assim criando uma comunidade gigante.

2.2 PARADIGMA

O Arduino tem sua própria linguagem de programação, ele é bem parecido com a linguagem C/C++, se você sabe algumas delas, já terá meio caminho andado. Ela contém funções próprias para dar o comando a componentes conectados a placa, você pode baixar bibliotecas de algum componente comprado, onde na sua documentação estará especificado o processo de instalação e suas respectivas funções. O Arduino conta também uma parte elétrica, os pinos (conexões de entrada) digitais e/ou analógicas que podem ser controladas por programação e outras entradas de alimentação, GND, 5v, 3.3v, etc. Importante usar uma protoboard para ter várias conexões em apenas um pino.

A linguagem Arduino é baseada em C / C ++, e qualquer pessoa que tenha estudado anteriormente C ou C ++ serão prontamente capazes de ler o código escrito. Se você não estudou antes, não se preocupe. A linguagem Arduino foi projetada para fazer a codificação como o mais indolor possível, permitindo que você se concentre no que pode ser feito, em vez de como isso é feito.

O Arduino se baseou muito na filosofia de projeto do *Processing*, que é ensinar os fundamentos da programação dentro de um contexto visual, e a equipe do Arduino, percebendo que precisava desenvolver uma linguagem que tornasse a prototipagem de ideias o mais simples possível, adotou uma filosofia semelhante. A decisão foi de usar a IDE do *Processing* como modelo para o IDE do Arduino, porque o sistema Arduino original era voltado para estudantes de arte e design que já estavam familiarizados com o *Processing*. Os vínculos próximos com o *Processing* ainda são evidentes hoje, e quaisquer melhorias feitas no *Processing* IDE podem ser importadas diretamente para o sistema Arduino.

A sua linguagem orientada a objeto, tipada e compilada. Essa linguagem faz instruções ao microcontrolador da sua placa Arduino, como ligar um led em um determinado momento. Para desenvolvedores mais experientes, o Arduino suporta a programação AVR por linguagem C. Esta linguagem apesar de ser bem mais rápida comparada a tradicional usando C++, por exemplo, um programa compilado com a linguagem nativa tem 928 bytes, já a programada em C custa 178 bytes, além de fazer uma comunicação *"low-level"*, fazendo instruções direto para o microcontrolador em binário. Mas você precisa de um nível de abstração maior e entender como funciona o microcontrolador da placa e outros assuntos.

2.3 PRINCIPAIS FOCOS DE APLICAÇÃO DA TECNOLOGIA

O foco do Arduino é controlar vários tipos de componentes de maneira inteligente, podendo auxiliar em serviços de produção e automação a nível doméstico, comercial ou móvel, como um CPL (controlador lógico programável).

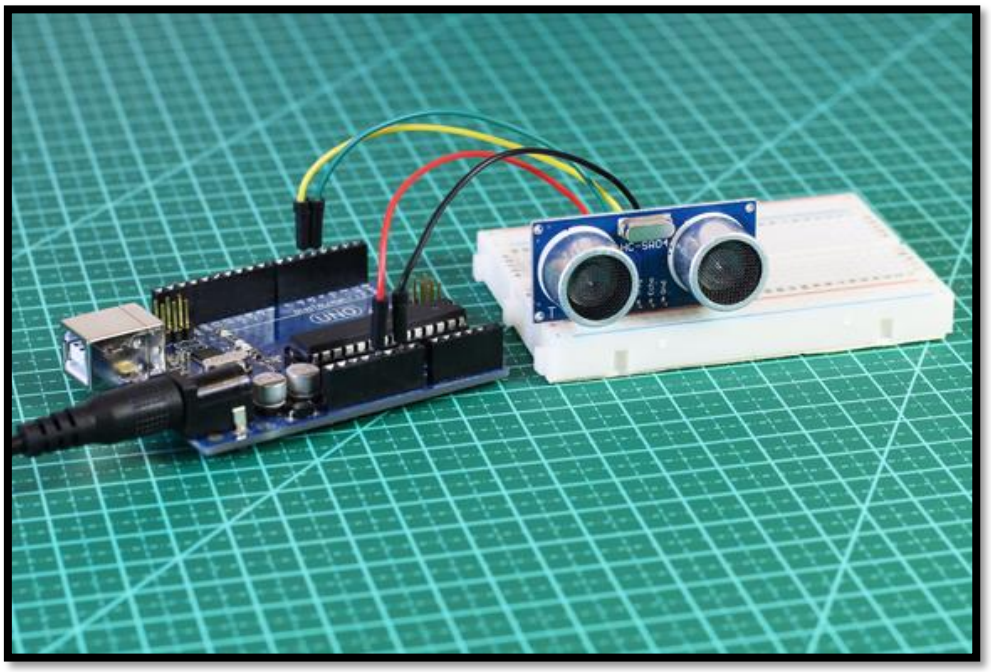
Têm vários projetos para ajudar no dia-a-dia, o mundo está na revolução das "Internet das Coisas", e com Arduino você pode fazer projetos inteligentes. Como um sistema de regar que aciona em tempos definidos; se tiver movimento em alguma área, emitir uma sinalização. Os projetos vão da sua imaginação.

O Arduino desafia industrias, por causa da sua facilidade de construções de sistemas. Comercialmente é utilizado uma tecnologia chamada de "sistemas embarcados" onde existem hardware e um software que o controla, especifico para uma aplicação, por exemplo, impressoras, micro-ondas, equipamentos hospitalares, ar condicionado, etc. Nesses sistemas existem microcontroladores, que tem capacidades computacionais. Isso não te lembra algo? O Arduino veio para simplificar e melhorar esses tipos de aplicações, uma pessoa facilmente pode criar uma aplicação sem entender afundo de eletrônica e programação. Os sistemas embarcados são complexos, mas eficazes, eles são criados por um propósito específico e otimizado para tal, com circuito elétrico impresso e compacto. Algo bem diferente ao Arduino, que tenta abranger e abstrair esses processos, para que seja capaz o desenvolvimento de milhares possibilidades de aplicações.

Na indústria não existem aplicações com Arduino, apesar de algumas exceções, ele não foi desenvolvido para aplicações industriais, para isso que temos os sistemas embarcados, o Arduino é mais utilizado para aplicações doméstica e estudo de eletrônica e programação. Apesar disso você pode se especializar em engenharia da computação, onde aprenderá bem mais afundo o funcionamento de um sistema embarcado e aplicações reais.

2.4 VANTAGENS, DESVANTAGENS E LIMTAÇÕES DA TECNOLOGIA

As suas vantagens foram decorridas a longo da apostila inteira, uma aplicação feita com Arduino é de fácil compreensão, já que nessa plataforma há um grande processo de abstração do processo ocorrido no micro controlador, com funções nativas. Além de ser modular, instalar outros pacotes de funcionalidades de outros programadores ao redor do mundo para sua aplicação é bastante fácil, com elas você pode ter acesso a outros componentes.



2.4 Sensor Ultrassônico - Felipe Flop

Você pode ver um sensor ultrassônico conectado ao Arduino, e para fazer projetos com um componente novo é bastante fácil, instalando sua biblioteca você

tem acesso as suas funções e pode o controlar de qualquer forma. Então qualquer pessoa do mundo com o conhecimento pode desenvolver e disponibilizar bibliotecas que auxiliam outros desenvolvedores a manipular componentes.

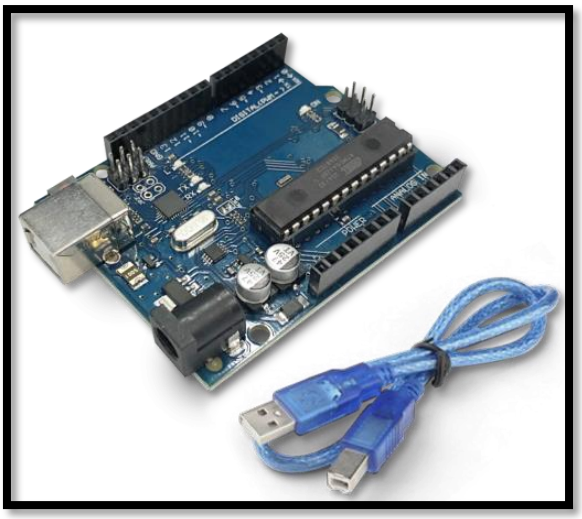
Agora suas desvantagens vêm do âmbito da performance e tamanho. O Arduino fica desvantajoso para o uso comercial e industrial, já com os sistemas embutidos são bem mais rápidos e preciso. Para uma aplicação mais complexa e maior, o Arduino se torna impraticável.

A sua limitação é a velocidade de resposta a comandos, o comando básico e muito utilizado `digitalWrite()` tem problemas de respostas, onde a instrução real na programação demora para acontecer. Isso é um grande empecilho para aplicações reais, onde se preza a rápida resposta. Outra limitação é o peso do programa compilado, onde em grandes códigos podem ter problema no seu carregamento. E por final, não haver uma forma de debugging do código, isso significa que seu programa tiver um erro não será mostrado para você, ficando mais complicado achar problemas recorrentes.

2.5 RECURSOS NECESSÁRIOS PARA USO DA TECNOLOGIA

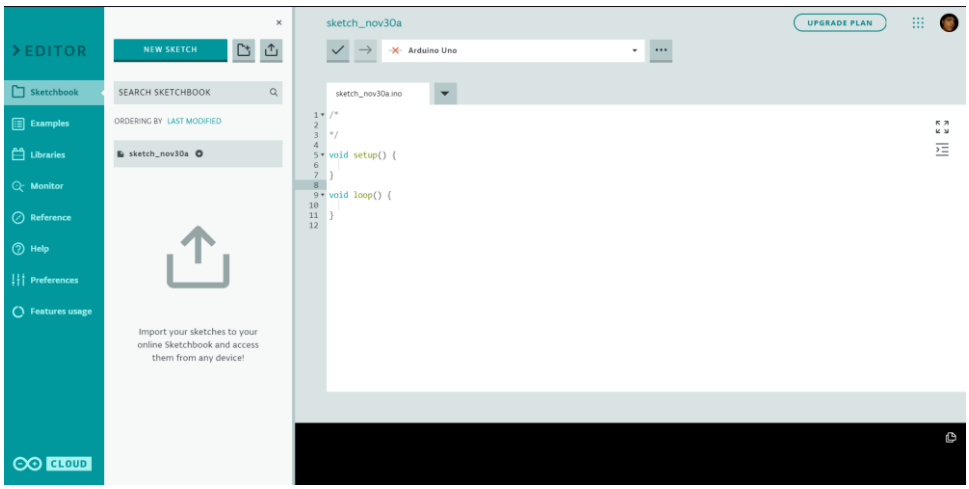
2.5.1 PREPARAÇÃO DO AMBIENTE FÍSICO

Para desenvolver na plataforma do Arduino você precisa de um modelo, para teste utilizaremos o Arduino Uno R3, que é um modelo acessível e genérico, mas você pode optar com modelos referentes a esse de outros fabricantes. No caso do desenvolvimento físico precisa-se de um computador/notebook com conexão usb. Primeiramente você deve conectar o cabo USB 2.0 A/B no Arduino e no seu desktop.



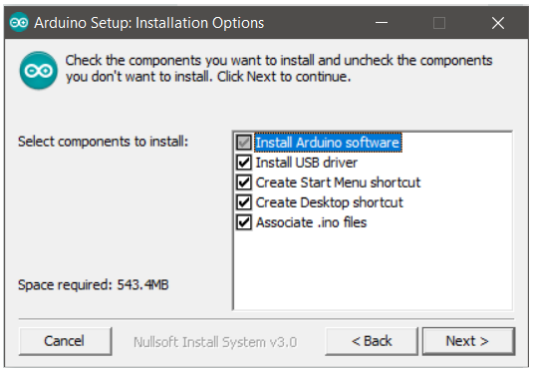
2.5 Imagem Arduino Uno R3 com cabo para conexão - cloudfront

Após isso, você tem que baixar uma IDE para começar o desenvolvimento, nisso você pode optar por um editor online (https://create.arduino.cc/editor/), mas nele você precisa estar conectado à internet.

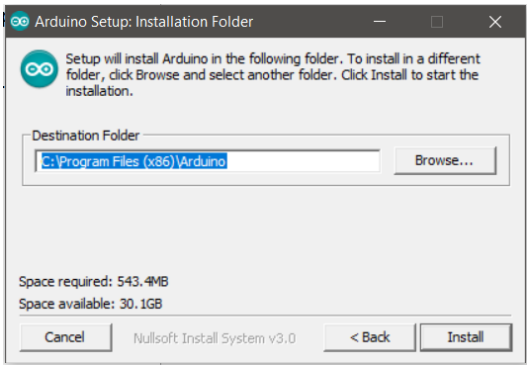


2.3 Imagem do editor online provido pelo Arduino – Acervo do autor

Depois do login você vai ser direcionado a essa tela com um breve guia, conectando seu Arduino ao computador pode-se fazer “sketch” e compilar os códigos. Agora temos o “Arduino Desktop IDE”, que é um aplicativo para computador, onde não depende de internet, para baixar entre no site do Arduino (https://www.arduino.cc/en/software) e escolha seu sistema operacional.

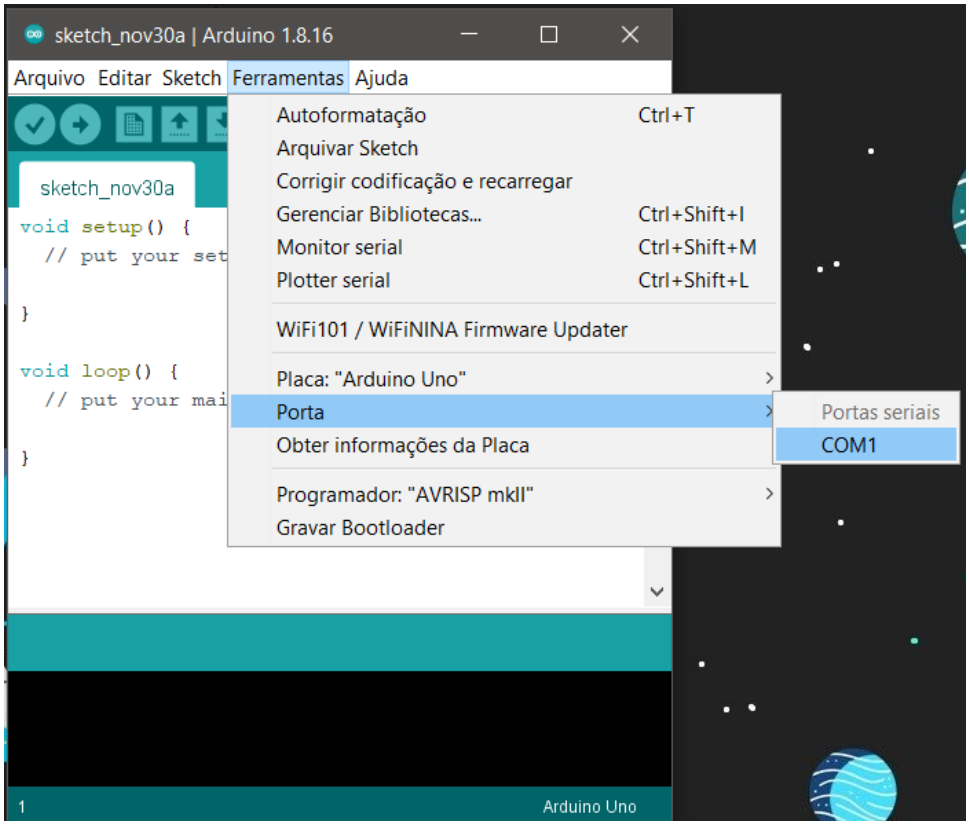


Clique em “next”;



Selecione uma pasta do seu computador em “Browser...” e depois clique em “install”. Após isso irá pedir a instalação de alguns plugins para a conexão do cabo do Arduino, apenas clique em “instalar”.

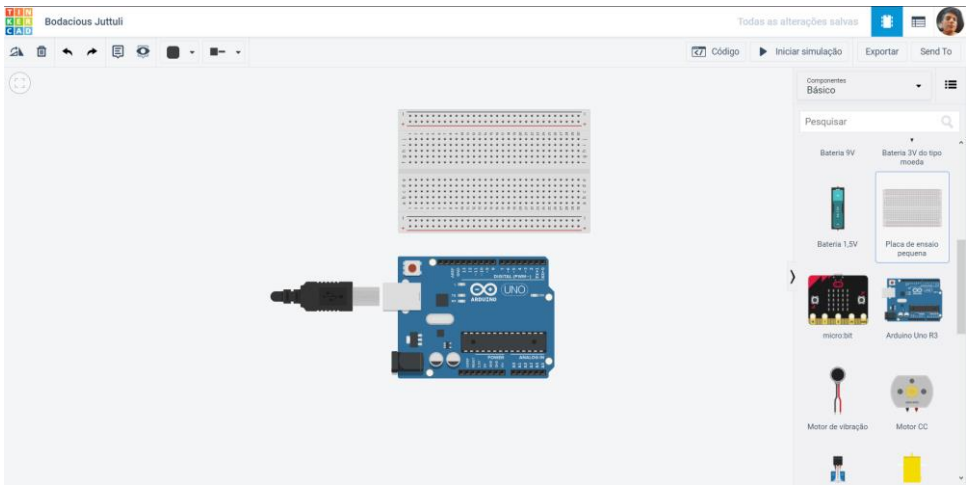
Posteriormente abra seu aplicativo Arduino e selecione: Ferramentas > Porta > Sua placa.



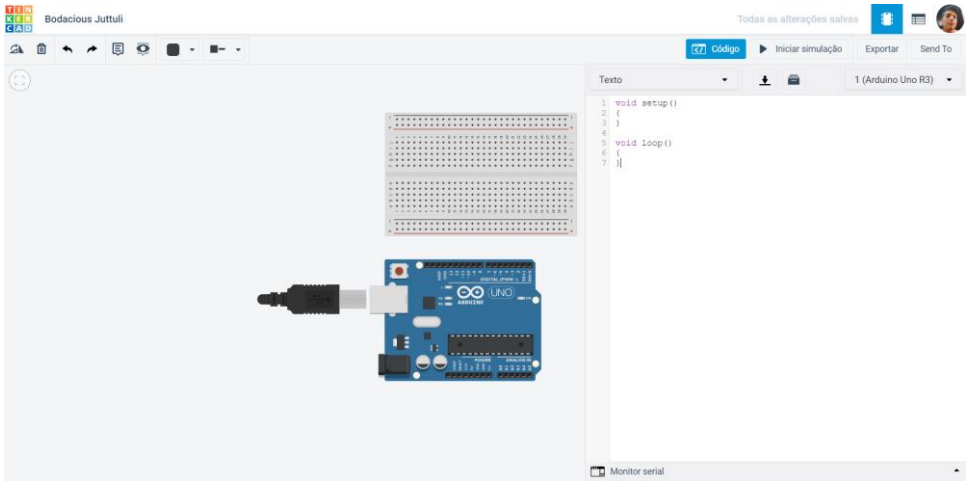
2.5.2 SIMULAÇÃO DE AMBIENTE ONLINE

Para simular um ambiente Arduino online utiliza-se o Tinkercad (https://www.tinkercad.com), utilizaremos essa plataforma. Para começar a desenvolver nisso é bem simples, crie um conta na sua página principal clicando em “inscreva-se agora”. Após isso você acessa uma página:

Depois clique em: Circuitos > Criar novo Circuito.



Agora crie esse pequeno esquema, clicando no “Arduino Uno R3” e “Placa de ensaio pequena”. Nessa aplicação de prototipação você pode pegar, arrastar e soltar componentes, também conectá-los com fios que estão disponíveis nos seus respectivos terminais. Para começar a desenvolver programas deve:



Clique em: Código > Selecione “Texto”. Apague tudo deixando o `void setup` e `void loop`. Agora está apto para entender o funcionamento do código e do Arduino nos próximos tópicos.

2.6 SINTAXE E SEMÂNTICA

A parte estrutural, sintática e semântica é muito aproveitada do C++, mas com funções próprias. Todos os códigos utilizados estão disponíveis nesse repositório (https://github.com/ifacLinguagem/Arduino-Ezequiel), para você copiar. Vamos começar com a entrada, processamento e saída de dados.

2.6.1 ENTRADA, PROCESSAMENTO E SAÍDA

Para esse tipo de procedimento temos 2 funções padrões para todo projeto com arduino, `void loop` e `void setup`:

```
// porta que o led está conectada é na porta 13
int pinoLed = 13;

// iniciando a função de setup do
// programa para definir propriedades
// de componentes
void setup()
{
  // explicaremos o pinMode depois
  pinMode(pinoLed, OUTPUT);
}
```

No início do programa estamos entrando com os dados, como em qual pino está conectado o nosso led, função `setup()` irá tratar de configurar nossos componentes após a entrada de dados, fazendo suas definições, isso se dá o processamento de dados.

Agora o `loop()`:

```
// irá sair os dados fazendo um loop infinito, enquanto toda a simulação essa função irá acabar e reiniciar, infinitamente
void loop()
{
```

```
  digitalWrite(pinoLed, HIGH);
}
```

Está função será ativada após a de setup, nela vamos ter a saída de dados, como o exemplo, o led irá ativar. Os “barra barra” (//) são como comentamos o código, não vai interferir no programa, é para comentar o que está se passando ou informações pertinentes.

2.6.2 TIPOS E DECLARAÇÃO DE VARIÁVEIS

A entrada de dados é onde vamos coletar a entrada de acordo com o componente, por exemplo, temos um componente, teremos que introduzir ele ao programa para poder o processar, isso se dar por meio de variáveis.

Variável é um tipo de memória disponível no nosso computador onde armazenamos algo para usar depois, ter uma referência desses dados para o manipular. Para isso devemos ter uma declaração, a linguagem do Arduino temos vários tipos de declaração. Segue a tabela com os tipos:

boolean	É um dado lógico, tem 2 tipos de atribuição, verdade (true) ou falso (false).
char	Para armazenar caracteres
byte	Carrega a informação de 1 byte, 8 bits
int	Números inteiros com 16 bits (-32768 a 32767)
unsigned int	Como o int, carrega números inteiros, menos os negativos (0 a 65535)
long	Também armazena números inteiros, mas com o dobro do armazenamento, 32 bits (-2147483648 a 2147483647)
unsigned long	Armazena números inteiros com 32 bits, mas sem números negativos (0 a 4294967295)
float	Este armazena também números decimais (ponto flutuante)
double	É igual o float, mas com dupla precisão
string	É uma série de caracteres, formando uma palavra
void	Tipo de variável vazio

Como vimos no exemplo anterior a variável `int` vou usada para armazenar o 13 como porta do led, que posteriormente o utilizamos para o ligar.

2.6.3 ESTRUTURA BÁSICA DE PROGRAMAÇÃO

A estrutura básica foi apresentada na entrada, processamento e saída de dados. Temos que definir o que está conectado no início do programa, tanto quanto bibliotecas de outros componentes, como um paineis LCDs. Depois temos que configurá-las, como um componente deve ser tratado, no caso do led é um dado de saída. A última é a que irá ficar repetindo eternamente no Arduino.

2.6.4 ESTRUTURA DE SELEÇÃO

Este tipo de estrutura irá fazer uma seleção lógica e condicional, na qual se validadas vão conter um código para ser executado, vamos simplificar com um pseudocódigo e depois converter para código:

```
/*
Se o pinoLed for igual a 13 > faça
    ligue o pinoLed
termine
Senão > faça
    desligue o pinoLed
termine
*/
```

Temos um pseudocódigo que faz uma condição, que se o pino do led, a variável que definimos lá em cima, for igual a 13, ele irá ligar o led, senão ele irá desligar o led. Isso é princípio da estrutura de seleção, onde terá uma condicional, se ela for satisfeita será feito uma instrução. Agora vamos passar para o código:

```
// início da condição com o if (se)
  if(pinoLed == 13) {
    // executa essa linha se verdade
     digitalWrite(pinoLed, HIGH);
  } else {
```

```
    // senão execute essa
    digitalWrite(pinoLed, LOW);
  }
```

O “se” dar lugar ao if, onde em parênteses temos as condições. Depois se cria um bloco com as chaves, onde se vai o código que será ativado se aquela condição for verdadeira. Agora temos o else, que será o “senão”, irá se ativar se a condição do if for falsa.

Nesse trecho vemos o “==”, que significa igualdade, ele compara o valor da variável `pinoLed` com 13, vendo se é igual, se sim a condição retorna *true*. Esse igual é um operador condicional, temos vários:

&&	Esse significa uma continuidade. Ex: se esta condição for verdade && (e) essa também
||	Esta é diferente da &&, apenas uma condição for verdadeira a expressão irá ser verdadeira. Ex: se esta condição for verdade || (ou) se essa também
==	Igualdade
!=	Diferença
!	Negação
>	Maior que
<	Menor que
>=	Maior ou igual que
<=	Menor ou igual que

2.6.5 ESTRUTURAS DE REPETIÇÃO

Estes são blocos que repetem se a uma condição for verdadeira. Exemplo:

`while(condição) {...}`. O *while* (enquanto), irá rodar um código se a sua condição for satisfeita, veja um exemplo:

```
void loop()
{
  // variável i de controle é definidada com 0
  int i = 0;
```

```
while(i < 3) { // enquanto o i for menor que 3 irá fazer
    digitalWrite(pinoLed, HIGH); // ligar o led do pinoLed
    delay(1000); // após 1 segundo (1000 milisegundos)
    digitalWrite(pinoLed, LOW); // desligando o led
    delay(1000); // mais um atraso de 1 segundo
    i = i + 1; // atribua o i somando-o com 1
  }

  delay(5000); // espere 5 segundos para repetir
}
```

O próprio `void loop` é uma repetição que não para. Temos outra estrutura de repetição, o for. O mesmo exemplo com essa outra estrutura:

```
void loop()
{
  for(int i = 0; i < 3; i++) {
    // cria a variável i atribuindo o 0, depois verifica se o i é
menor que 3 e depois que executa do código incrementa o i com mais 1
    digitalWrite(led, HIGH);
    delay(1000);
    digitalWrite(led, LOW);
    delay(1000);
  }

  delay(5000);
}
```

Nesse trecho tem o mesmo efeito que a outra repetição, o “for” funciona nesse princípio, primeiro você carrega uma variável, iniciando com 0. Depois a verificação que será os números de contagens que o led irá piscar. Se a variável passar por essa

condição ativa o código que está entre as chaves. Após o código ser executado, a variável i é incrementada com 1, assim voltando ao loop até a variável não satisfazer a condição.

2.6.6 ESTRUTURAS HOMOGÊNEAS DE DADOS

Este tipo de estrutura permite que armazenamos vários tipos de variáveis do mesmo. São exemplos vetores e *strings*. Para você declarar um *array* (vetor) tem que:

```
int meuVetor[]; // iniciando um meuVetor inteiro vazio, se dar com os dois colchetes.
int pinos[4] = {13, 12, 11, 10}; // agora inicia uma matriz "pinos" com 4 posições, começa a contar do 0, e depois esse array foi alimentado com números de pinos conectados
```

Vamos ver um exemplo real, as vezes temos que conectar e gerenciar vários componentes do mesmo tipo, podemos agrupá-los com um vetor:

```
int pinos[4] = {13, 12, 11, 10};

void setup()
{
  for (int i = 0; i <= 3; i++)
  {
    pinMode(pinos[i], OUTPUT);
  }
}


void loop()
{
  for (int i = 0; i <= 3; i++)
  {
    digitalWrite(pinos[i], HIGH);
  }
```

```
}
```

Nas duas funções percorremos o vetor por meio de uma repetição e configuramos e depois ligamos cada um dos led que estão no pino 13, 12, 11 e 10. Assim evitando repetir o mesmo código para fazer a mesma função. Importante salientar que o array é contato apartir do 0, então se você tiver um array com 2 posições, para o referenciar precisa-se: `array[0]` e `array[1]`

Também temos outro tipo de estrutura homogêneas, *strings*. Quando *strings* são usadas como uma matriz de char, normalmente terminam com um nulo caractere ASCII 0. Isso permite que outras funções reconheçam quando chegaram ao final da *string*. Uma matriz char pode ser declarada de várias maneiras. Todos os seguintes são válidos:

```
char minhaString [10];
char minhaString [3] = {'O', 'l', 'á'};
char minhaString [6] = {'O', 'l', 'á', '\ 0'};
char minhaString [] = "Olá";
char minhaString [3] = "Olá";
char minhaString [10] = "Olá";
```

A primeira das linhas de código anteriores configura uma matriz não inicializada, a segunda linha anexa automaticamente `'\0'` ao final e a terceira linha o inclui. O quarto linha divide automaticamente os caracteres e dimensiona automaticamente a matriz, a quinta linha divide automaticamente a matriz, e a sexta linha deixa espaço na matriz para uma *string* maior

2.6.7 FUNÇÕES

Por meio desses exemplos vimos bastante o uso de funções, elas servem para como uma sequência de código que podem ser reutilizáveis. No começo já nos deparamos com `setup()` e `loop()` que são chamadas automaticamente quando o programa se inicia. Para criar sua própria função deve-se especificar como será o retorno, por exemplo:

```
int soma() {
```

```
  return 1 + 1;
}
int dois = soma();
```

Se inicia uma função tipando o seu retorno, como a soma resulta em um valor inteiro, a função é do tipo `int`. Após tipar o retorno deve-se dar um nome a função, abra parênteses e abra um escopo, como uma estrutura. Depois o retorno dessa função soma, é atribuído em uma variável “dois”, para invocar uma função devemos apenas abrir e fechar os parênteses.

```
int soma(int num1, int num2) {
  return num1 + num2;
}
int resultado = soma(2, 2)
```

Vamos dar o próximo passo, parâmetro da função. No caso da nossa função temos 2 parâmetros, um inteiro `num1` e `num2`; após isso retorna o resultado da soma dos valores passados. Para chamar uma função devemos passar 2 argumentos, os números inteiros para realizar a soma, atribuindo o resultado (4) na variável resultado.

```
int pinos[4] = {13, 12, 11, 10};


void setup()
{
  for (int i = 0; i <= 3; i++) {
    pinMode(pinos[i], OUTPUT);
  }
}


void piscarLed(int pino, int tempo) {
  digitalWrite(pino, HIGH);
```

```
  delay(tempo);
  digitalWrite(pino, LOW);
  delay(tempo);
}



void loop()
{
  noInterrupts();
  piscarLed(pinos[0], 10000);
  piscarLed(pinos[1], 2000);
  piscarLed(pinos[2], 500);
  piscarLed(pinos[3], 6000);
}
```

Neste exemplo temos uma função chamada de “piscarLed”, onde é definido uma sequência de ações com um pino que está o led conectado e o tempo do intervalo que o mesmo ficará piscando, assim reutilizando o mesmo código, impedindo repetições. O `noInterrupts()` significa que os outros delays dos leds não irá interferir no loop, assim não interrompendo o progresso do código e o delay só interferindo na frequência do próprio led. Lembre-se, se você estiver repetindo o mesmo trecho de código, pense se não pode fazer uma função.

2.6.8 FUNÇÕES NATIVAS

No decorrer desta apostila vimos vários tipos de funções nativas, quando usamos o `delay()`, onde é uma função que não tivemos que programar, então são funções nativas providas do Arduino, auxiliando o desenvolvimento sem que precisarmos os programar. O Arduino simplifica muitas tarefas usando funções prontamente acessíveis para controlar a entrada e saída de dados digital e analógica, bem como matemática, trigonometria e funções de tempo

É uma caixa de ferramentas que é dado a nós para utilizarmos na programação do no algoritmo, onde não precisamos saber como elas foram feitas, mas sim seu efeito. O `delay()` pausa o programa por um tempo passado em milissegundos no argumento da função.

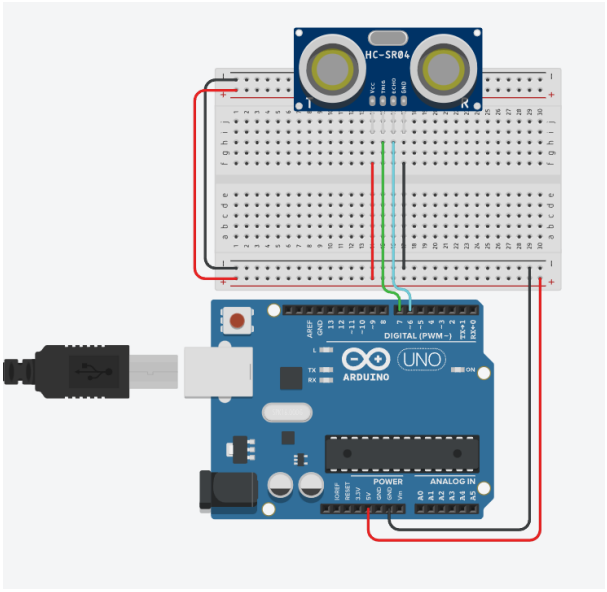
O Arduino provê milhares funções para o desenvolvimento, a tabela abaixo cita as principais, mas você pode ler a documentação oficial (https://www.arduino.cc/reference/en/#functions) e estudar qual delas servi para o seu projeto.

digitalRead	Identifica o valor do pino digital, se ele está HIGH (ligado) ou LOW (desligado)
digitalWrite	Nele você especifica qual pino digital irá ligar ou desligar.
pinMode	Especifica um pino para se comportar como como uma entrada (INPUT) ou saída (OUTPUT)
random	Gera um número aleatório podendo conter um máximo e mínimo

2.7 SUGESTÃO DE USO

Com Arduino existem infinitos projetos que você pode desenvolver, tenha uma ideia e bote ela em prática, pesquise componentes requeridos e estude a lógica por traz.

Irei apresentar um mini robô que desvia de obstáculos, que poderá montar um pequeno carrinho, mas fixarei a parte eletrônica e programação. Primeiro a montagem eletrônica na placa.



2.7 Imagem primeira montagem com sensor ultrassônico

Utiliza uma placa de ensaio para ser flexível as conexões, sem ser preciso soldar direto no componente. O componente referente é um sensor ultrassônico (HR-SR04) para medir a distância de uma área curta. O primeiro e último terminal são 5v e GND (neutro) respectivamente para sua alimentação, posteriormente é conectado em 2 portas digitais do Arduino, 7 e 6, para coletar informações que o sensor ultrassônico está medindo.

Agora a parte da programação, para um exemplo físico é preciso a instalação de uma biblioteca específica, mas o tikercad já é nativo e vamos apenas o considerar.

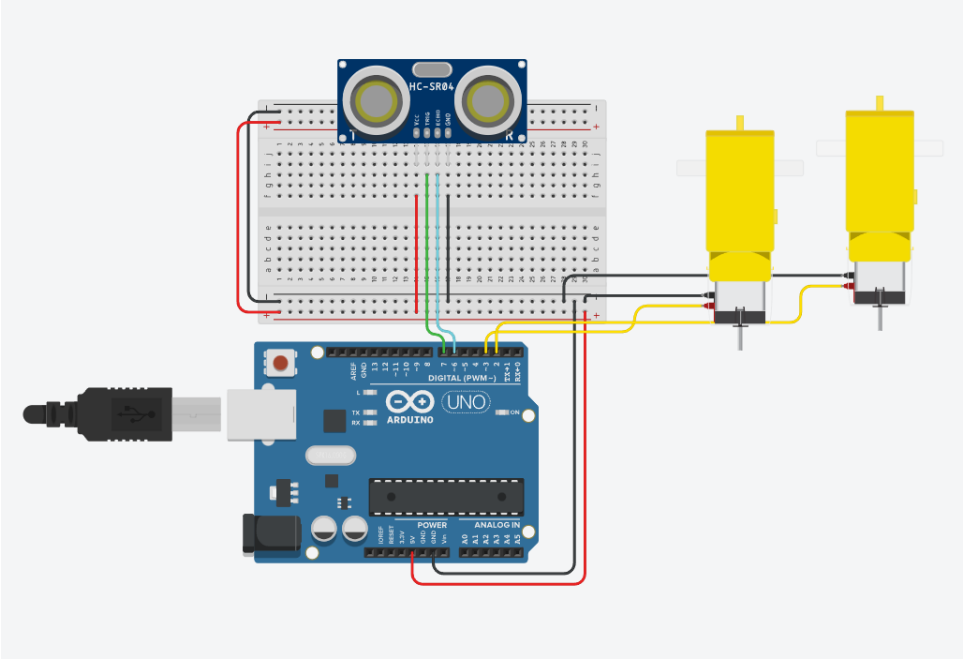
```
int PinoTrigger = 7; // Pino que irá transmitir a distancia
int PinoEcho = 6; // Pina que irá receber a distancia

int duracao = 0;
int distancia = 0;

void setup()
{
  // Configurando o tipo de dados transmitido pelos pinos digitais
  pinMode(PinoTrigger, OUTPUT);
```

```
  pinMode(PinoEcho, INPUT);


  Serial.begin(9600);
  // definindo a taxa de bits do serial para 9600
}


void loop()
{
  // Ligando o pino que transmite a distancia
  digitalWrite(PinoTrigger, HIGH);
  // Ele dura 10us (microsegundos) para calcular a distância de cada pulso
  delayMicroseconds(10);
  // após esse intervalo o desligamos
  digitalWrite(PinoTrigger, LOW);


  // O pulseIn captura a duração de um pulso no pino que recebe a informação do mesmo
  duracao = pulseIn(PinoEcho, HIGH); // A variável duração carrega essa duração
  // devemos passar para centímetros
  distancia = duracao * 0.017175;
  // depois printamos no serial da ida para verificar a distância em cm
  Serial.print(distancia);
  Serial.println("cm");
  delay(100);
}
```

A fórmula para transformar a leitura do pino de entrada que chega em metros por segundos é a seguinte: Deve-se transformar a velocidade que o sensor percorre no ar (343,5 m/s) para centímetros, depois dividir por 1 milhão, que é o equivalente a 10us. O resultado é 0,03435, mas o sensor ultrassônico recebe o sinal duplicado,

porque ele mede a duração da ida e volta de cada pulso, assim deve-se dividir esse resultado por 2: 0,03435 / 2 = 0,017175. Agora para saber a distância que o sensor ultrassônico mediu é multiplicar ele pela nossa constante.

Agora a parte dos motores:



O polo negativo é ligado no GNU do Arduino e seu polo positivo é ligado em um terminal 3 e 2 da placa, para os gerenciar. Para começarmos precisamos que acrescentar:

```
// Pino dos 2 motores
int motor1 = 3;
int motor2 = 2;

void setup()
{
  // Definindo o modo do pino dos motores
  pinMode(motor1, OUTPUT);
  pinMode(motor2, OUTPUT);
}
```

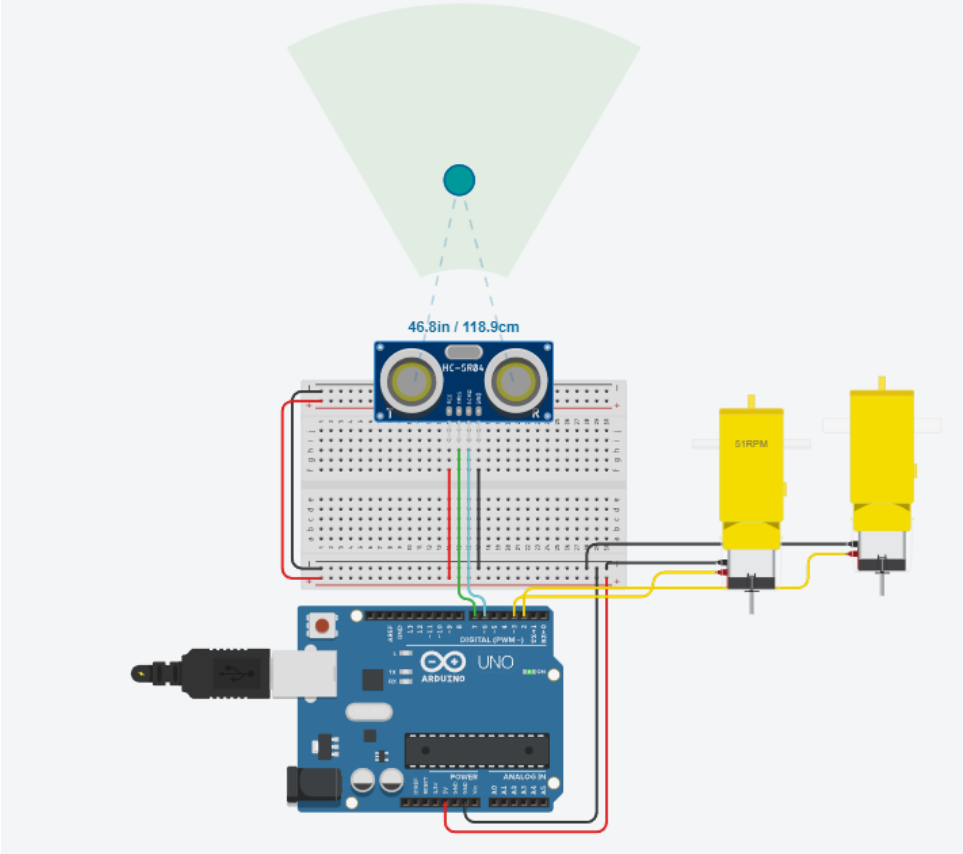
Definindo os modos dos motores para saída, vamos apenar desligar e ligar os motores, após devemos organizar a lógica do nosso programa:

```
Início programa

Ligar o sensor ultrassônico

Deligar e ler a distância do sensor ultrassônico

Se a distância do sensor for muito pequena

Desligue um motor

Ligue o outro motor

Senão

Ligue os dois motores

Fim do programa
```

Este vai ser o nosso pseudocódigo para a ideia da nossa aplicação, quando o sensor ultrassônico medir uma distância curta, um dos motores irá parar e o outro irá ligar, assim fazendo uma curva. Como ficará no código:

```
// Verificando a distância medida se ela for menor ou igual a 150
  if(distancia <= 150) {
    // ligando um motor e desligando outro, fazendo uma curva
     digitalWrite(motor1, HIGH);
    digitalWrite(motor2, LOW);
  } else {
    // senão ele apenas irá andar reto
    digitalWrite(motor1, HIGH);
    digitalWrite(motor2, HIGH);
  }
```

Agora o nosso programa está feito, um protótipo de um carrinho que desvia de obstáculos avistados.


46.8in / 118.9cm
HC-SR04
51RPM
UNO
ARDUINO

3 CONCLUSÃO

Agora você tem uma boa ferramenta para criar inúmeros projetos, o Arduino tem uma comunidade incrível desbrave-as e descubras pessoas com mesmos interesses para compartilhar conhecimento.

Essa apostila fui um grande desafio em ser feita, tem pouco conteúdo oficial sobre a linguagem própria do Arduino, fiz a pesquisa por meios de livros e documentações oficiais, também em sites especializados. Comecei no Arduino com um projeto de robótica, quando vi o poder que tenho em o usar fiquei apaixonado em aprender mais e mais, de programá-lo em fazer uma instrução que exerce algo incrível. Fiz vários micros-projetos, como um semáforo com botão para fechar o sinal para o pedestre atravessar, e várias ideias de aplicações que desafiavam a lógica de programação e a eletrônica.

O que eu indico é mão no Arduino e no código, se você se interessar por programação, que foi o lado que eu segui, ou pelo caminho da eletrônica, são ótimas opções. O Arduino tem vários benefícios te engaja a compreender esse mundo melhor, contudo não te limita a ele, apesar de ser simples.

REFERENCIA

JOHN. **Story and History of Development of Arduino.** [S.l.: s.n., 2021]. Disponivel em: <https://www.circuitstoday.com/story-and-history-of-development-of-arduino>. Acesso em: 1/12/2021

Arduino. **Arduino Playground.** Arduino CC, 2018. Disponível em: <https://playground.arduino.cc/Portugues/HomePage/>. Acesso em: 2/12/2021.

Arduino. **Documentação de Referência da Linguagem Arduino.** Arduino CC, 2021. Disponível em: <https://www.arduino.cc/reference/pt/>. Acesso em: 2/12/2021.

Arduino. **What is Arduino?** Arduino CC, 2021. Disponível em: <https://www.arduino.cc/en/Guide/Introduction>. Acesso em: 2/12/2021.

Arduino. **Install the Arduino Software (IDE) on Windows PCs**. Arduino CC, 2021. Disponível em: <https://www.arduino.cc/en/Guide/Windows#download-the-arduino-software-ide>. Acesso em: 2/12/2021.

Brauer Di Renna, RBDR. Et al. **Introdução ao kit de desenvolvimento Arduino**. Edição A2021M06D10. Niterói - RJ: Programa de Educação Tutorial – PET, junho de 2021.

Daniel Zancan, Marcos D.Z. **Controladores Programáveis**. Santa Maria - RS: Colégio Técnico Industrial, 2011.

Souza, Fábio. **O Arduino está se tornando o novo padrão da indústria?** Embarcados, 2019. Disponível em: <https://www.embarcados.com.br/o-arduino-esta-se-tornando-o-padrao-da-industria/>. Acesso em: 5/12/2021.

Fernandes dos Anjos, Alexandre. **Performance do digitalWrite() no Arduino**. Embarcados, 2019. Disponível em: <https://www.embarcados.com.br/performance-do-digitalwrite-no-arduino/>. Acesso em: 5/12/2021.

Igor Borçatti, Pedro. **Preparando o Eclipse para microcontroladores AVR**. Embarcados, 2018. Disponível em: <https://www.embarcados.com.br/eclipse-para-microcontroladores-avr/>. Acesso em: 5/12/2021.

Wiring. **Wiring**. Disponível em: <http://wiring.org.co>. Acesso em: 6/12/2021.

Rodrigo Toste Gomes a.k.a. Et al. **Programação em C no AVR**. Senso, 21/12/2010.

Gedeane Kenshima**. Nas linhas do Arduino: Programação Wiring para não programadores**. Ucrânia: Novatec Editora, 25 de março de 2020.

Martin Evans, Joshua Noble, Jordan Hochenbaum**. Arduino em Ação**. Brasil: Novatec Editora, 29 de maio de 2013.