Arduino + Brino para a robótica educacional

Gabriel Rodrigues Pacheco, Giulia Fricke Galice, Mateus Berardo de Souza Terra, Rafael Mascarenhas Dal Moro, Victor Rodrigues Pacheco.

Brasília, 2016

Índice

1. Introdução

A tecnologia está presente em todos os aspectos da vida humana, desde rastreadores de sono e outras tecnologias vestíveis até ferramentas educacionais e os mais tradicionais computadores e smartphones. Dentro desse contexto, o desenvolvimento de habilidades específicas se torna cada vez mais importante, mesmo para quem não trabalha na área de TI, ao ponto de, em 2020, saber uma linguagem de programação ser tão fundamental quanto conhecer o inglês.

Uma linguagem de programação é uma série de instruções, com regras sintáticas e semânticas, que são executadas por um processador. Estas são, em sua maioria, feitas utilizando o inglês como base. Observando a dificuldade de vários jovens e adultos com a língua, nós criamos uma linguagem de programação em português para o Arduino, o Brino, que almeja facilitar o entendimento da lógica de programação, sendo caracterizada por ser intuitiva.

O Arduino é uma placa com um micro controlador que pode ser utilizada para prototipagem de sistemas e máquinas de forma fácil e rápida. Não apenas ao hardware, mas também a um conjunto de software se refere o nome. A placa é baseada em um processador Atmel AVR e é feita em hardware livre. Portanto, é possível acessar a página do Arduino e baixar os esquemáticos da placa para montar o seu próprio clone.

Tal plataforma é atualmente utilizada em diversas áreas. Não apenas no ramo da tecnologia e desenvolvimento, mas também por pessoas adeptas do movimento maker (Faça Você Mesmo, do inglês DIY - Do It Yourself), que adotam esta placa pela curva de aprendizagem relativamente pequena. Além de hobbistas, artistas estão aproveitando as possibilidades dela para construir obras interativas e responsivas.

A versatilidade do Arduino fez com que ele se tornasse muito popular. Nosso primeiro contato aconteceu no ano de 2014. Desde lá, desenvolvemos diversos projetos: uns voltados para IoT (Internet of Things - Internet das Coisas), alguns jogos e até robôs autônomos. Para aprender, utilizamos alguns livros e muita internet! O próprio fórum do Arduino é um ótimo local para buscar auxílio assim como outros pela WEB. Sites como o *instructables* também são uma mina de ouro para quem busca um projeto interessante e não sabe por onde começar.

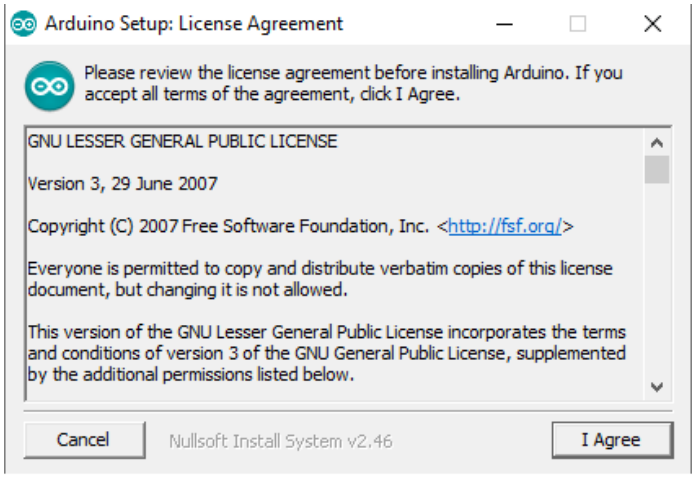
Os projetos disponíveis na internet são programados utilizando a própria linguagem do Arduino. Se você está utilizando o Brino, pode buscar nosso auxílio por e-mail, na página do Facebook ou até utilizar o código nativo do Arduino em seu rascunho (o Brino oferece suporte ao código nativo do Arduino). Tudo bem, você já leu uma folha inteira de teoria sobre o que é o Arduino, linguagens de programação, etc.... Se você, como nós, tem um espírito maker, já está de saco cheio de texto e quer começar a desenvolver seu projeto e fazer seu Arduino ser útil, então vamos pôr a mão na massa! Antes disso, uma pequena dica: não seja levado completamente por sua impulsividade, aproveite um tempo antes de cada projeto para elaborar um bom planejamento e avaliar as possibilidades, isso evitará que perca tempo e dinheiro.

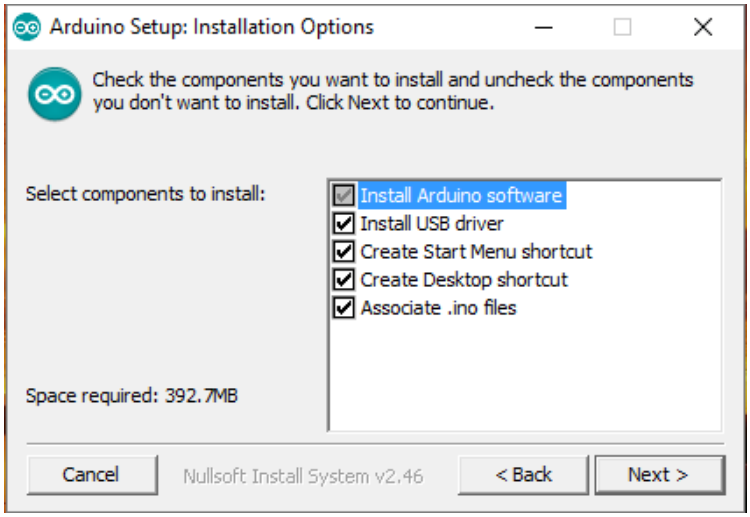
2. Começando

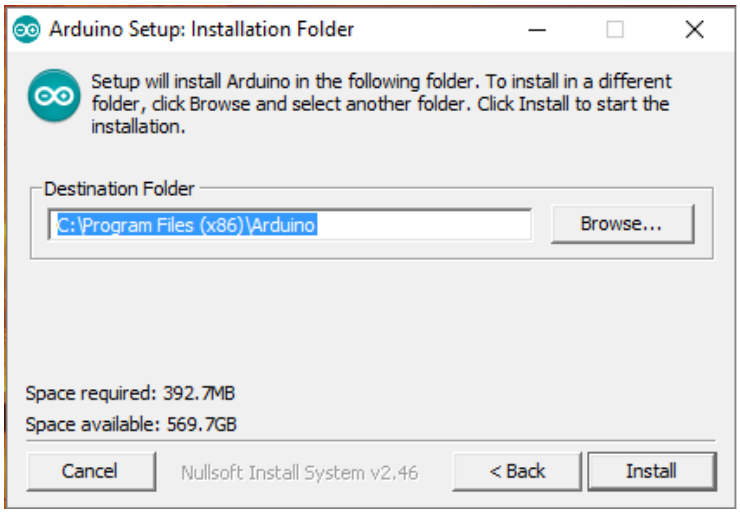
2.1 Arduino:

Para começar você vai precisar, claro, de um Arduino físico ou de uma versão virtual funcional dele. Seja um original ou um clone, a placa que você preferir provavelmente funcionará. Nessa apostila utilizaremos o Arduino Nano como referência. Recomendamos que o resto dos componentes seja adquirido quando você começar a desenvolver o projeto do capítulo, uma vez que comprar todos juntos pode ser custoso. Instruções de confecção de placas de circuito impresso, soldagem e outras habilidades necessárias para o desenvolvimento de alguns projetos, podem ser encontradas ao final do livro.

Primeiro, vamos configurar o seu computador para que você possa programar sua placa. Entre no site do Arduino [*arduino.cc*] e vá na aba downloads para baixar a IDE mais recente, que na ocasião era a versão 1.6.7. Siga as imagens para completar a instalação:



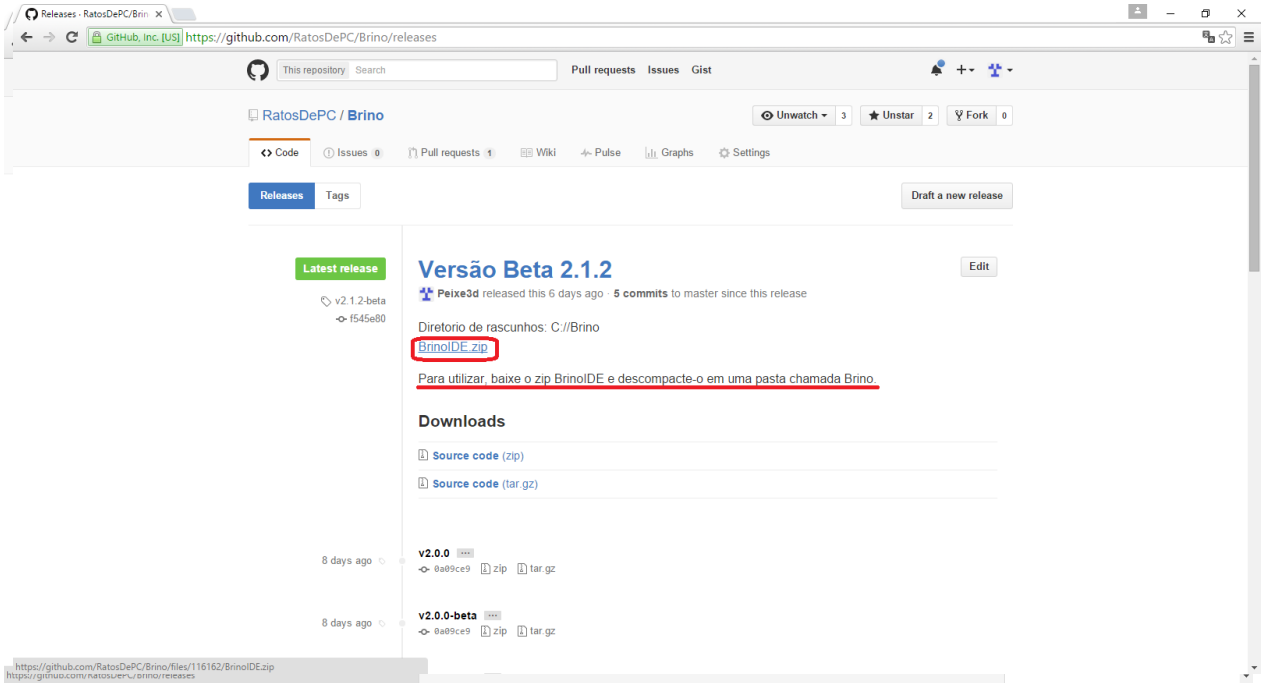


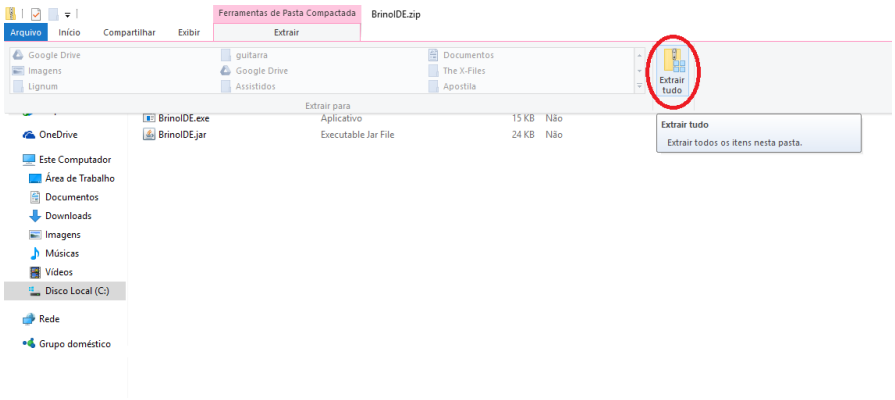


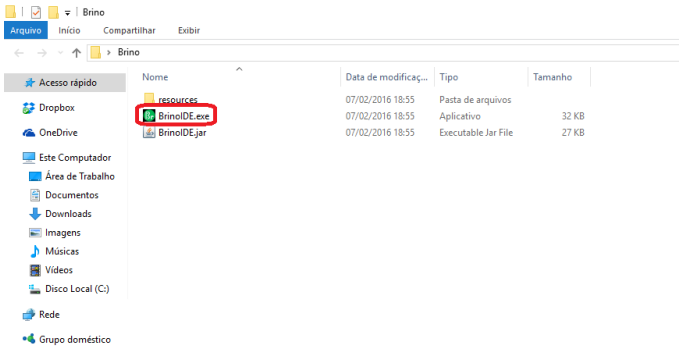
2.2. Brino:

Depois de instalar a IDE do Arduino, você está pronto para instalar o Brino. Para isso, basta acessar a página do GitHub: *http://ratosdepc.github.io/Brino*; na área de primeiros passos, você encontrará o link para a página de releases e os nossos contatos no final. Já na página de releases, baixe o zip da versão mais recente e descompacte-o em algum lugar (de preferência onde você possa encontrá-lo facilmente). Depois disso, basta abrir o arquivo BrinoIDE.exe e você estará pronto para começar! Caso queira consultar algum código de exemplo, estes podem ser facilmente encontrados na pasta exemplos. Siga as imagens abaixo para completar a instalação.

Nota: Quando for compilar algum código, não se esqueça de selecionar a placa que você está utilizando na aba de ferramentas/placa.







2.2.Fritzing:

Fritzing é um programa que busca tornar a eletrônica e a programação acessível a todos. É uma ferramenta intuitiva e rápida para quem deseja documentar, digitalmente, projetos feitos com Arduino. Ele possui diversas opções para a montagem de circuitos sobre uma protoboard virtual, disponibilizando inúmeros modelos de Arduino para o usuário. Além disso, o Fritzing permite exportar esquemáticos, pode ser usado como IDE, entre outras funções.

Para baixar, acesse *http://fritzing.org/home/* e, na parte superior, clique em downloads e busque pela versão mais recente compatível com o seu sistema operacional. Uma vez obtida a pasta .zip, basta extraí-la para uma pasta convencional. Dentro de tal pasta encontra-se o arquivo fritzing.exe, que é o programa.

2.3.Pronto para começar:

Depois de instalar tudo, é hora de desenvolver seu primeiro projeto: “Piscar”. Esse programa é extremamente simples, não necessita de experiência prévia, ou de qualquer outro componente além da sua placa Arduino, pois ela já contém um LED que pode ser controlado. Entretanto, se você nunca programou antes, sugerimos que você leia o capítulo de Introdução à Programação, localizado na área de habilidades adicionais.

Nota: àqueles que já possuem alguma experiência com programação, recomendamos ler os tipos de variáveis da linguagem Brino e as funções e métodos obrigatórios em todo rascunho.

3. Introdução à Programação

Para criar nossos próprios códigos é necessário aprender palavras-chaves e suas estruturas. Assim como um texto em língua Portuguesa ou em inglês, programas de computador utilizam

linguagens com regras específicas de sintaxe para que o computador possa compreender o que queremos que ele faça.
Nesse capítulo abordaremos assuntos como:

- Variáveis;
- Comentários;
- Incrementadores;
- Instruções e laços de controle;
- Operadores lógicos;
- Funções ou Métodos.

3.1 Variáveis:

Variáveis são muito usadas na programação, pois elas são capazes de armazenar dados. Uma das formas mais simples de pensar em uma variável é como uma caixa ou balde onde o computador pode armazenar ou ler dados. Elas tornam o código mais fácil de se entender e de ser mantido. Por exemplo, se você possuir uma variável chamada *pinoLED* que define o pino 13 como uma saída para um LED e, mais a frente, decidir usar o pino 8 para isso, será fácil a troca, sendo necessário apenas substituir o valor 13 pelo valor 8 na declaração do valor da sua variável.

Esse exemplo poderia ser aplicado no segmento a seguir:

```
Numero pinoLED = 13;          <= Aqui está a nossa variável!
Configuracao() {
        Pino.definirModo(pinoLED, Saida);
}
Principal() {
    Pino.ligar(Digital.pinoLED);              // liga a porta digital acedendo o LED
    esperar(1000);                            // espera por um segundo
    Pino.desligar(Digital.pinoLED);           // desliga a porta digital apagando o LED
    esperar(1000);                            // espera por um segundo
}
```

Variáveis são divididas em locais e globais, dependendo da parte do código em que ela for declarada. As locais são aquelas declaradas dentro de uma função (Principal(), para(), if(), etc.), enquanto as globais são declaradas fora delas. Variáveis locais só podem ser usadas dentro de suas funções, já as globais podem ser usadas em qualquer parte do programa.

Nota: É possível criar duas variáveis locais com o mesmo nome em funções diferentes, mas tome cuidado para não se esquecer que elas serão independentes entre si.

Agora que entendemos qual a utilidade das variáveis em nossos programas vamos examinar os seus tipos:

- Numero: As variáveis do tipo Numero (deve ser escrito dessa forma para ser entendido pelo compilador, sem acento e com letra maiúscula. Isso se aplica a diversas palavras que serão abordadas no capítulo) são muito usadas pois são capazes de, como o próprio nome sugere, armazenar números inteiros entre
  -32.768 a 32.767, ou seja, um número de 16 bits.
  Ex.: Numero minhaVariavel = 3600;

- NumeroDecimal: Esse tipo de variável é capaz de armazenar números de até 32 bits com um ponto decimal.
  Ex.: NumeroDecimal raio = 3,5;

- Letra: Essa variável armazena um caractere ASCII (iremos abordar isso mais a frente), ou seja, ela é capaz de armazenar qualquer caractere (a, A, 1, &, entre outros). Operações aritméticas podem ser aplicadas sobre esses dados. Seu dado deve vir entre aspas simples (‘ ‘).
  Ex.: Letra nota = ‘A’;

- Palavra: Esse tipo especial de variável pode ser comparado a uma serie de caracteres. Ela é usada para armazenar palavras e frases. Seu dado deve vir entre aspas simples ou duplas (‘’ ou “”).
  Ex.: Palavra saudacao = ‘oi’;

- Condicao: A menor variável que vamos estudar é usada para guardar apenas dois valores, Verdadeiro ou Falso, e será muito usada em operações lógicas e como controle.
  Ex.: Condicao chovendo = Falso;

Outro ponto importante quanto as variáveis são os nomes que elas podem receber. Salvo as palavras chave predefinidas pela linguagem de programação vigente, elas podem ter qualquer nome. Mesmo com toda essa liberdade, recomendamos fortemente que não usem acentos ou caracteres especiais, como ç, @, etc. Além de preferir, sempre que possível, nomes sugestivos (não, variável_123 não é nada sugestivo) para que o código possa ser mais facilmente entendido pelos outros e por você mesmo. Tais nomes devem começar com uma letra ou uma underline. São exemplos de bons nomes: valorSensor, motorDireito, porta_LED e leituraDistancia.

Para declarar uma variável como identificador de uma porta analógica, adicione um “A” maiúsculo à sua frente. Ex.: Asensor.

Caso se deseje trabalhar com uma constante, a palavra-chave Constante pode ser adicionada ao início da variável, tornando-a invariável.

Ex.: *Constante Numero minhaVariavel = 3600;*

Nota: Com os tipos de números inteiros, você pode obter uma situação chamada *roll over*, em que um valor é somado ou subtraído extrapolando os limites da variável fazendo com que o fim de um intervalo role para outra extremidade. Por exemplo, temos uma variável do tipo Numero com o valor armazenado de 32.767 e somamos 1 a ela. O valor resultante dessa operação será de -32.768 e não 32.768.

3.2 Comentários:

Comentários são um recurso muito utilizado na programação, uma vez que, por meio de "notas do autor", contribuem para um melhor entendimento e organização do código. Na hora da compilação, essas linhas são ignoradas pela máquina, fato que não desmerece a relevância dessa ferramenta. Enquanto trabalham em algoritmos extensos e complexos, é de suma importância que os programadores deixem notas explicando o que está sendo feito e qual o objetivo de cada bloco. Tal processo dinamiza o desenvolvimento. Imagine procurar um bug, ou consertar um erro em um código com 500 linhas (marco não muito difícil de ser alcançado) sem ter a menor ideia de onde procurá-lo...

No Brino, os comentários podem ser introduzidos por meio de uma barra dupla na frente da linha (//) fazendo com que essa linha em específico seja um comentário. Outra forma de usar essa ferramenta é utilizar uma barra acompanhada por um asterisco (/*) abrindo um bloco de comentário que só terminará em um asterisco seguido por uma barra (*/).

Nota: evite colocar acentos em um comentário e em todo o código no geral. Ao abrir o código em diferentes editores de texto, eles podem não entender um caractere acentuado da mesma forma. De forma simples, nem todos os editores de texto "escrevem" na mesma língua, por isso entendem os acentos de forma diferente.

Ex.: *// Isso e um comentario.*

Ex. 2:
```
/*
 * Isso e um comentario de bloco
 */
```

3.3 Incrementadores:

No Brino existem alguns operadores que podem incrementar ou decrementar o valor de uma variável. Além daqueles a que estamos acostumados (+, -, *, /), há também incrementadores especiais. São eles:

Operador	Exemplo	Resultado
++	Variável++;	O valor da variável vai ser incrementado em uma unidade
--	Variável--;	O valor da variável vai ser decrementado em uma unidade

+=	Variável += n;	O valor da variável será incrementado em n unidades (no caso de Palavras, o trecho n será adicionado ao final)
-=	Variável -= n;	O valor da variável será decrementado em n unidades
*=	Variável *= n;	O valor da variável será igual ao produto do seu antigo valor e n
/=	Variável /= n;	O valor da variável será igual ao quociente do seu antigo valor e n

Nota: Quando estamos falando de programação os símbolos da divisão e da multiplicação costumam ser substituídos por uma barra simples (/) e por um asterisco (*) respectivamente.

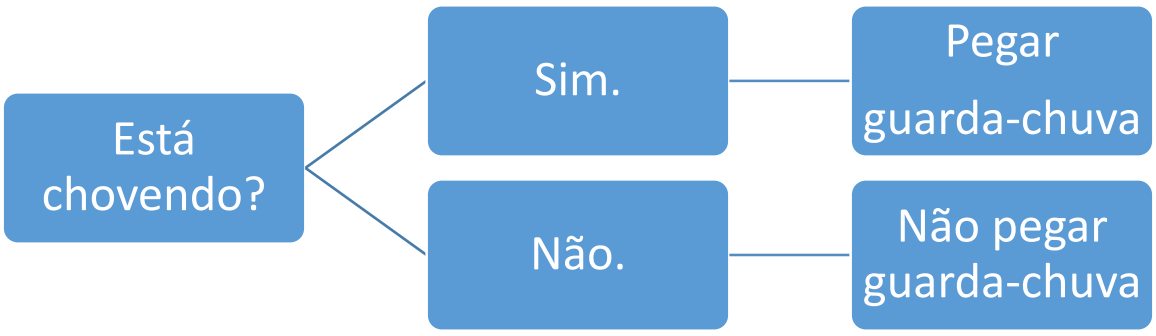
Nota: Quando atribuímos o valor por meio de uma igualdade (=), o valor da direita é atribuído ao lado esquerdo.

Ex.: X = 2 * 5

3.4 Instruções e laços de controle:

Assim como nós, as máquinas são capazes de tomar decisões, porém de maneira muito mais simples. Os códigos fazem decisões por meio das chamadas operações booleanas. Essas operações aceitam apenas dois resultados, o verdadeiro e o falso. Uma forma de pensar quanto a isso é relacionar esse recurso a situações do cotidiano. Por exemplo, se estiver chovendo, devo pegar o guarda-chuva, caso contrário, não preciso pegá-lo. Aqui, a expressão booleana seria o fato de estar ou não chovendo e a instrução de controle seria pegar ou não o guarda-chuva.

Uma forma muito usada para representar essas situações é por meio de diagramas de blocos como o representado a seguir:



Agora que temos uma noção do que se trata, podemos estudar os Operadores Relacionais. Eles nada mais são do que comparadores que usaremos para analisar informações tendo uma saída de Verdadeiro ou Falso. Os operadores estão representados na tabela a seguir:

Operador	Descrição	Exemplo	Resultado
>	Maior que	2 > 1	Verdadeiro
>=	Maior ou igual a	2 >= 2	Verdadeiro
<	Menor que	2 < 1	Falso
<=	Menor ou igual a	2 <= 2	Verdadeiro
==	Igualdade	A == A	Verdadeiro
!=	Desigualdade	A != A	Falso

Assim como podemos observar os operadores não estão limitados a comparar apenas números, mas também podem comparar Letras e outras variáveis.

Nota: Tome cuidado!!! Não se esqueça que o símbolo = faz uma atribuição, sendo diferente do ==, que faz uma comparação.

Entendendo como obter resultados Verdadeiros e Falsos podemos desenvolver um pouco mais a ideia usando os operadores condicionais. Esses são os operadores que vamos usar para tomar decisões quando associados aos comparadores. Veremos a seguir como usar os operadores se, senao e o senao se.

Eles seguem as estruturas exemplificadas abaixo:

```
se(expressão booleana){
        // Se a expressão for Verdadeira esse bloco será executado.
}
// Se ela for Falsa o bloco será ignorado.
```

```
se(expressão booleana){
        // Se a expressão for Verdadeira esse bloco será executado.
}
senao{
        // Se ela for Falsa esse bloco será executado.
}
```

```
se(expressão booleana){
        // Se a expressão for Verdadeira esse bloco será executado.
}
senao se(outra expressão booleana){
        // Se a segunda expressão for Verdadeira
        // esse bloco será executado.
```

```
}
senao{
        // Se nenhuma dos blocos anteriores forem executados,
        // esse será.
}
```

Além do *se* e do *senao*, existe a instrução enquanto(). O laço enquanto(), como o próprio nome sugere, executa um bloco de código enquanto uma condição, entre os seus parênteses, for verdadeira. Ele é usado para realizar um processo enquanto for necessário, enquanto a expressão for verdadeira, e para quando não for mais necessário, a expressão se tornar falsa.

```
enquanto(Condicao){
        // Esse bloco é repetido enquanto a condição for verdadeira.
}
```

O laço para() é utilizado para repetir um determinado bloco de código um número determinado de vezes usando, para isso, uma variável como contador. Ao contrário dos outros anteriormente citados, o para() aceita mais parâmetros. O primeiro é a declaração das variáveis locais, sendo seguido pela expressão booleana e por uma expressão de incremento ou decremento do valor da variável.

```
para(Tipo <nome> = <valor>; <nome> <operador> <valorReferencia>; <incremento>){
    // Bloco que será repetido.
}
```

Exemplo:

```
para(Numero x = 0; x <= 10; x++){
    //Bloco a ser repetido;
}
```

3.5 Operadores lógicos:

Os operadores lógicos são usados quando uma expressão booleana não é o suficiente para a tomada de decisões, então, por meio deles, nós podemos ter mais do que uma expressão booleana com apenas uma saída. Um jeito interessante de pensar nesses problemas é voltando ao exemplo do guarda-chuva. Eu estou saindo de casa, se estiver ensolarado eu não irei pegar o guarda-chuva, mas caso esteja chovendo ou pareça que vai chover, devo pegá-lo. A seguir estão representados os operadores:

Operador	Descrição	Exemplo	Resultado
E	Se ambas forem verdadeiras	2 > 1 e 2>=2	Verdadeiro

Ou	Se uma das duas for verdadeira	2 >= 2 ou 1>3	Verdadeiro

Repare nos exemplos, para saber a saída, devemos fazer essa operação por etapas. No caso *2 > 1 e 2>=2*, temos duas expressões booleanas. Como já sabemos resolvê-las podemos dizer que o problema fica Verdadeiro e Verdadeiro. Observa-se que ambas as expressões têm uma saída verdadeira e estão ligadas pelo operador *e*, logo, o resultado será Verdadeiro.

3.6 Funções ou Métodos:

Funções ou métodos consistem basicamente em um determinado bloco de código escrito pelo desenvolvedor para evitar repetir um conjunto de instruções. O Brino requer o uso de pelo menos duas funções. Essa ferramenta é extremamente importante para reduzir o tamanho dos códigos e otimizar a utilização da memória do computador. Para se declarar uma função, é necessário dizer que tipo de dado ela nos retornará, ou seja, qual a resposta que ela nos fornece após efetuar todas as suas instruções, que pode ser qualquer tipo de variável ou *SemRetorno* se ela não responde nada e seu nome. Além do retorno, precisamos declarar os dados, ou argumentos, que ela receberá para efetuar suas operações. A declaração de uma função segue o modelo:

```
TipoDeRetorno <nome>(TipoDeVariável <nome_argumento_1>, ..., argumento n){
    // Bloco de instruções da função ou método.
}
```

O Brino possui duas funções obrigatórias que iremos debater:

- Configuracao():

Executada uma vez quando o Arduino é inicializado. É responsável por preparar o hardware para a execução do loop principal. O método de Configuracao() é executado uma única vez quando o Arduino é ligado e é ignorado até que seja reiniciado. É nele que definimos a conexão com a porta USB, os modos de operação dos pinos, entre outros parâmetros.

```
Configuracao(){
// Esse bloco é repetido apenas uma vez na inicialização.
}
```

- Principal():

O método Principal() é um dos mais usados para se colocar a parte principal do programa. Ele é executado a partir do momento que o Arduino é ligado, após a execução da configuração, até o momento em que ele é desligado podendo ser repetido incontáveis vezes. Nele colocamos todas as operações necessárias para o funcionamento contínuo do projeto.

```
Principal(){
// Esse bloco é repetido continuamente.
}
```

4 Introdução a eletrônica básica

Uma parte muito importante da robótica é a eletrônica, ou o hardware, pois ela que é capaz de interagir com o meio externo. É por meio de atuadores (motores e servomotores) e sensores que ele será capaz de analisar e se locomover lidando com o ambiente à sua volta. Esse é um assunto muito amplo e que pode requerer muitos cálculos, mas como esse material possui caráter apenas introdutório, só abordaremos as partes realmente necessárias como:

- Grandezas;
    - Corrente Elétrica (I);
    - Resistência (R);
    - Tensão (U);
- Componentes:
    - Resistores;
        - LDR;
        - Potenciômetro;
    - Buzzer;
    - Interruptores;
        - Relés;
    - Capacitores;
        - Cerâmica;
        - Eletrolítico;
    - Diodos;
        - LEDs;
    - Motores
        - CC simples;
        - Servo;
        - De passo;
    - Baterias
- Associação de componentes;

4.1 Grandezas:

Eletrônica é o ramo da física estuda as propriedades e aplicações de mecanismos cujo funcionamento baseia-se do movimento de elétrons. Nesta unidade veremos as principais grandezas pertencentes à essa ciência.

4.1.1 Corrente Elétrica (I):

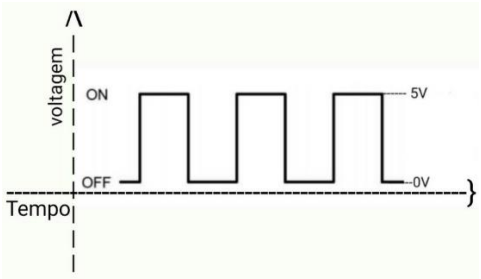
A corrente elétrica é o fluxo ordenado de cargas por um condutor, ou seja, para os nossos estudos ela pode ser definida o fluxo de elétrons por materiais como fios e componentes. Ela ocorre entre dois pontos com potenciais diferentes e pode ser medida em Ampere (A). Uma forma de pensar nela é comparando-a com a água. A corrente seria a quantidade de água que passa pelo cano, e o cano o seu condutor. A corrente pode ser um dos seguintes tipos:

4.1.1.1 Contínua (C.C.):

A corrente contínua ocorre quando o fluxo das cargas ocorre sempre na mesma direção, ou seja, quando os polos se mantêm constantes. Ela é constituída pelos polos positivo e negativo e é usada em baterias e pilhas. Abaixo, vê-se um gráfico do valor corrente em relação com o tempo:

PWM:

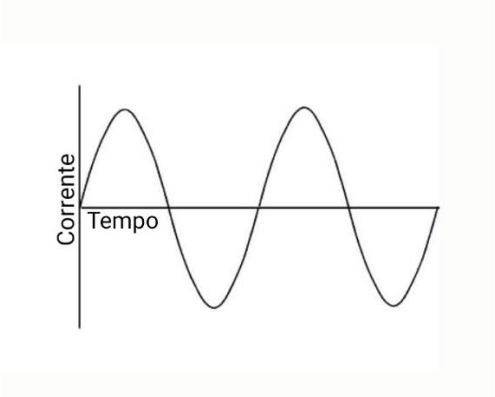
PWM (Pulse-Width Modulation) significa modulação por largura de pulso e pode ser usado para fazer comunicações ou para controlar o valor da alimentação. Ele consiste em ligar e desligar várias vezes consecutivas uma voltagem em um curto período de tempo, fazendo com que aparentasse de que se usou um valor intermediário. É utilizado no controle de alguns motores.



4.1.1.2 Corrente Alternada (C.A.):

A corrente alternada, diferentemente da contínua, altera o seu sentido com o decorrer com o tempo, fazendo com que as cargas fiquem indo e vindo. Ela é composta por fases e, muitas vezes, pelo fio neutro. Ela possui menor perda quando comparada à corrente contínua e, por isso, é usada principalmente em linhas de transmissões e tomadas. A sua forma de onda mais comum é a

senoidal (representada abaixo), por ser a mais eficiente. Mas, em certas aplicações, outras formas podem ser utilizadas.



4.1.3 Resistência (R):

A resistência elétrica é a capacidade de um corpo qualquer de se opor a corrente elétrica mesmo quando uma diferença de potencial é aplicada. Na analogia com a água, a resistência está diretamente relacionada a espessura do cano, permitindo que passe mais ou menos líquido. Ela é medida em ohm (Ω) e está diretamente relacionada à lei de Ohm, que será explicada mais a frente.

4.1.4 Tensão (U):

A tensão, também conhecida como DDP (Diferença De Potencial) ou voltagem, assim como o nome já sugere, representa a diferença de potencial entre dois pontos, ou seja, é a quantidade de energia gerada para movimentar uma carga elétrica. A tensão é medida em Volt (V) ,em homenagem ao físico italiano Alessandro Volta, um jeito de pensar nela é como a força que "empurra" as cargas. Na comparação com a hidráulica, ela seria a diferença de altura de duas caixas d'água, que gera uma energia potencial nas moléculas.

4.1.4.1 Polos Elétricos

A ideia de polos elétricos está diretamente relacionada com a tensão. Há dois principais polos na corrente contínua, o positivo (+) e o negativo (-), que no nosso caso será o ground (GND). O negativo se caracteriza como o polo de menor potencial elétrico (V-), que no Arduino possui 0 volts. Já o positivo, é o polo de maior potencial elétrico que no Arduino é normalmente de 5 ou 3,3V. A voltagem de um circuito pode ser medida a partir da diferença desses 2 polos pela fórmula:

U= (V+) – (V-)

4.2 Componentes:

4.2.3 Resistores:

Um resistor é um componente que apresenta uma certa dificuldade ou resistência à passagem de corrente. Os resistores são amplamente utilizados em circuitos, seja para regular a tensão ou a corrente. A resistência de um resistor é medida em ohms. Para identificar os diferentes

resistores de acordo com sua resistência, utilizam-se listras coloridas que seguem um código convencionado.

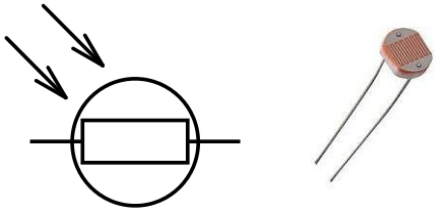
Nota: A tabela está localizada no capítulo 7.2 para consulta.

Os resistores podem ser associados para obter valores não disponíveis comercialmente, ou específicos para o projeto. Esse processo está explicado no capítulo de associações.

Além dos resistores comuns, existem resistores que variam a sua resistência de acordo com fenômenos físicos como a luminosidade, no caso do LDR, ou a temperatura, no caso do Termistor. Devido à suas propriedades, tais resistores são muito utilizados como sensores na robótica. Depois de alguns capítulos, você perceberá que grande parte dos sensores são resistivos. Agora falaremos um pouco sobre o LDR e sobre o potenciômetro, os resistores variáveis mais utilizados:

4.2.1.1 LDR:

Um LDR (Light Dependent Resistor) é um Resistor Dependente de Luz ou Fotorresistor. Ele tem a capacidade de variar a sua resistência em função da luz que incide sobre si. Tipicamente, à medida que a intensidade da luz aumenta, a sua resistência diminui, permitindo que mais corrente flua por ele. O LDR pode ser usado para confecção de sensores de luminosidade. Abaixo podemos ver a imagem de um LDR e seu símbolo esquemático.

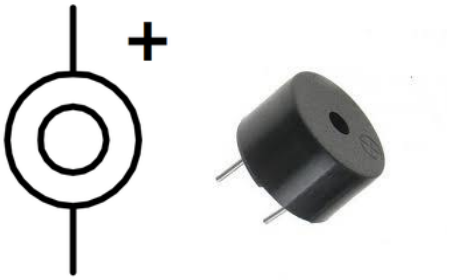


4.2.1.2 Potenciômetro:

Esse tipo de resistor possui sua resistência interna que varia em função de uma parte mecânica, ou seja, à medida que seu eixo gira, ele altera sua resistência. Tal característica o torna muito útil em interações homem/máquina ou como sensor para definir a posição de uma parte mecânica. Sua simbologia em esquemáticos está representada a seguir (à esquerda):

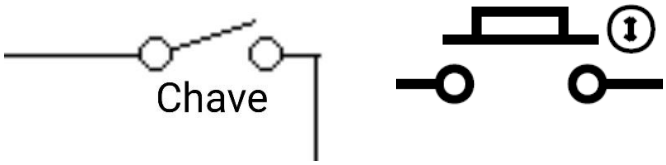
4.2.2 Buzzer:

Um buzzer é um componente capaz de produzir sons na frequência recebida atuando de maneira semelhante a uma caixa de som, porém com um consumo menor. Ele possui polaridade definida (a perna mais longa é o positivo) e é composto 2 camadas de metal e uma camada interna de cristal piezoeléctrico.



4.2.3 Interruptores:

Componente responsável por manter o circuito aberto ou fechado, quando acionado, fecha o circuito e permite a passagem de corrente.

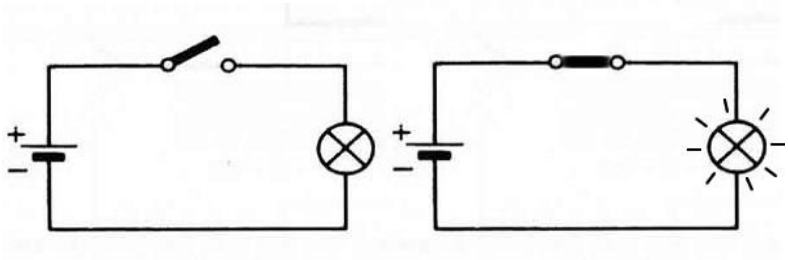


Circuito aberto/fechado

Uma das maneiras mais simples de controlar um circuito elétrico é o ligando e desligando. Para isso pode ser utilizado um interruptor que abre e fecha o circuito, ou seja, permite ou não a passagem de corrente. O circuito aberto bloqueia a passagem da corrente, ou seja, existe uma descontinuidade no "caminho" da energia.

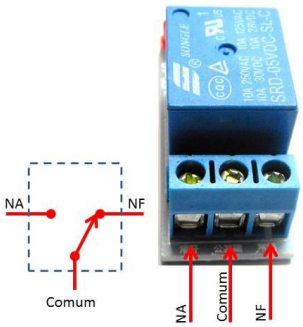
Suponha que você tenha uma lanterna, e obviamente, você não deseja que ela fique apenas ligada ou desligada, mas sim que essa mudança possa ocorrer facilmente. Para resolver esse problema existe o botão dela, o nosso interruptor, que é capaz de fechar ou abrir o circuito. Para exemplificar esse problema temos os esquemáticos de uma "lanterna" simples abaixo, em que o

da esquerda representa o circuito aberto (sem a passagem da corrente e, consequente mente, sem brilho na lâmpada) e o da direita o circuito fechado (com a lâmpada brilhante).



4.2.3.1 Relés:

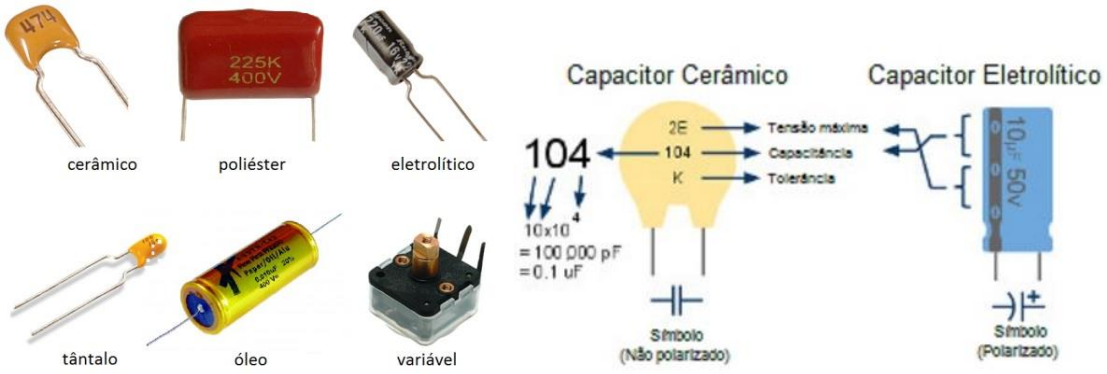
Um relé é um interruptor eletromecânico que é acionado por uma corrente elétrica mudando a posição da chave, assim permitindo ou não a passagem de uma corrente elétrica. Ele possui diversas aplicações, com um destaque especial no uso de baixas correntes para o controle de correntes maiores como, por exemplo, na automação residencial e no controle de portas de elevadores.



4.2.4 Capacitores:

Capacitores são dispositivos capazes de armazenar energia elétrica na forma de campo eletroestático. Tal habilidade, chamada capacitância, é qualificada de acordo com a quantidade de carga armazenada. A unidade de medida é o Farad. A seguir discutiremos dois dos tipos de capacitores, que são os mais utilizados em pequenos projetos:

Nota: Capacitores carregados apresentam resistência infinita em circuitos de corrente contínua.



4.2.4.1 Capacitor de cerâmica:

Também conhecidos como capacitores cerâmicos, destacam-se como os capacitores mais utilizados atualmente. Presentes desde circuitos de corrente contínua (C.C.), até circuitos de altas frequências. Não possui polaridade, ou seja, funciona nos dois sentidos.

Nota: O valor dele é tabelado. Para encontrá-lo use o código escrito nele.

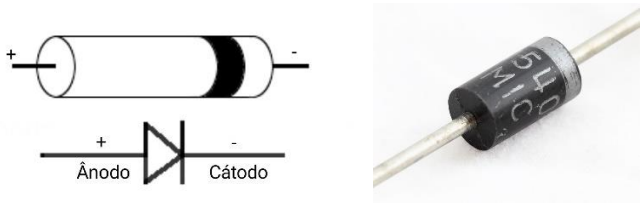
4.2.4.2 Capacitor eletrolítico:

Não recomendados para projetos que envolvam sinais de frequências elevadas, sendo para esses mais recomendados outros capacitores. Possuem destaque para conexões entre circuitos e filtragem de sinais de baixa frequência e possui polaridade (faixa e perna maior).

4.2.5 Diodos:

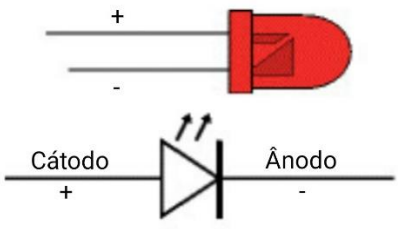
O diodo é um componente de dois terminais que conduz a corrente em apenas um sentido, bloqueando a sua passagem no sentido oposto. Ele é um semicondutor muito usado em retificadores e em circuitos de proteção. Existem diversos modelos, com diversas funções.

Nota: seus uso geralmente implica em uma pequena queda de tensão (0,7 V).



4.2.5.1 LEDs:

Um LED (Light Emitting Diode) é um diodo emissor de luz. Isso significa que ele tem as propriedades de um diodo (descrito acima) e é capaz de liberar luz própria, como uma pequena lâmpada. Ele é um semicondutor e a sua simbologia está representada a seguir junto a uma imagem identificando os seus polos:



Nota: O valor de tensão e corrente de cada LED pode ser consultado no capítulo 9, na parte de LEDs.

RGBs:

LEDs RGBs são basicamente três LEDs associados, sendo um deles vermelho, um verde e um azul. Com esse componente podemos emitir luzes de cores distintas, basta regular a intensidade de cada LED presente no RGB. Este possui quatro pernas sendo a mais longa o cátodo ou o ânodo comum.

4.2.6 **Motores**

Nesse capítulo iremos abordar os motores elétricos, dispositivos capazes de transformar energia elétrica em mecânica e vice-versa. Eles podem ser classificados como atuadores e são um componente que deve ser escolhido com calma. A seguir veremos os principais tipos de motores C.C..

- Motores C.C. simples;
- Servo motores;
- Motores de passo.

4.2.6.1 Motores C.C. simples:

Os motores “normais” de corrente continua possuem baixa potência e alta velocidade, tornando necessário, o uso de uma caixa de redução, que nada mais é do que um conjunto de engrenagens que mudam a velocidade e a potência a mesma proporção. O controle desse tipo motor com o arduino normalmente requer uma ponte H.

4.2.6.2 Servo motores:

Servo motores são motores de corrente contínua com uma caixa de redução acoplada e um circuito controlador. Existem dois tipos de servo motores:

Nota: Fique atento quanto ao consumo deles, eles podem precisar de uma alimentação externa.

Rotação limitada:

Esse é o tipo mais comum de servo motor. Sabendo o ângulo em que ele se encontra e, por meio de um sinal, pode alterar a sua posição para o ângulo desejado. Esse tipo de motor usa uma porta PWM e geralmente tem sua movimentação em ângulos menores que 360°.

Rotação contínua:

Esse tipo de motor não possui nenhum limite de angulação, porém não é possível controlar o ângulo deles, mas sim o seu sentido de rotação e velocidade.

Nota: Ele pode precisar de calibração, que pode ser feita por meio de um orifício em sua lateral.

4.2.6.3 Motores de passo:

Os motores de passo são usados para realizações de movimentos muito precisos, utilizado para isso “passos”, que seriam pequenos movimentos do motor. Ele utiliza uma quantidade variada de fios que podem acionar as suas bobinas internas, possibilitando assim, controlar a sua posição por meio de padrões. São comumente encontrados, por exemplo, em impressoras.

Nota: Existem placas especiais para facilitar o seu controle.

4.2.7 Baterias

Um ponto muito importante no planejamento de um robô é a escolha de suas baterias. Pilhas são energia química armazenada que pode ser convertida facilmente em energia elétrica, enquanto baterias são conjuntos de pilhas ligadas entre si. Baterias podem influenciar muito no preço e no peso do robô e, quando mal escolhidas, podem causar diversos problemas incluindo a parada completa do robô. Os pontos mais importantes na escolha delas são:

- Corrente;
- Tensão;
- Peso;
- Custo;

Antes que possamos falar sobre baterias, deve-se ter em mente o efeito memória, popularmente conhecido como “bateria viciada”. Ele ocorre principalmente nos modelos mais antigos, chamados Níquel Cádmio, e geralmente quando não ocorre a descarga completa antes da

recarga ou caso seu carregamento seja interrompido antes de estar completo. Esse mau uso faz com que a bateria “se acostume” a receber apenas uma parte da sua carga, reduzindo sua duração.

Outro ponto que deve ser destacado é que as baterias sofrem um desgaste natural com o passar do tempo e algumas precisam ser armazenadas de maneiras específicas para que possam ser utilizadas por um longo período de tempo. A seguir iremos falar dos tipos de baterias mais comuns, citando as suas principais vantagens e desvantagens.

Nota: Fique atento a forma correta de descarte de baterias, que deve ser feito em lugares específicos de coletas, pois elas podem ser muito agressivas ao meio ambiente

Nota: Vale a pena lembrar que as pilhas e baterias do mesmo tipo podem ser associadas com o fim de obter maior tensão ou maior corrente/autonomia. O capítulo 4.3 aborda esse tema.

Níquel Cádmio (NiCa):

Esse tipo de bateria já reinou o mercado entre os anos de 1990 e 2000, porém foram substituídas por modelos mais eficientes. Elas possuem um baixo custo e um peso médio, mas sofrem, quase sempre, com o efeito memória. Outra desvantagem desse tipo de bateria é a sua grande agressividade ao meio ambiente.

Hidreto metálico de níquel (Ni-MH):

Essas baterias são muitos parecidas com a anterior, porém não sofrem tanto com o efeito memória e possuem maior carga por peso. Elas também são menos agressivas ao meio ambiente.

Chumbo-Ácido:

As baterias de chumbo são muito eficientes quando se trata de armazenar uma grande carga, porém são muito pesadas. Elas são muito utilizadas atualmente em automóveis e em empilhadeiras. As baterias de chumbo utilizadas em carros possuem 12V de tensão e correntes de até 70ª.

Íons de lítio (Li-Ion):

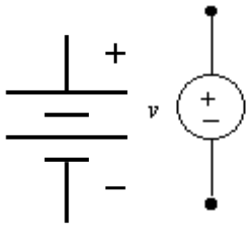
Essas baterias são modernas e não sofrem com o efeito memória. Estão presentes em quase todos celulares e aparelhos portáteis atuais por serem leves e poderem armazenar grandes quantidades de energia. Elas possuem diversos tamanhos e as apresentadas abaixo possuem tensão média de 3,7V. Uma desvantagem é o seu custo que é mais elevado que os dos modelos anteriormente citados.

Polímeros de Lítio (Li-Po):

Muitos as consideram as mais avançadas do mercado atual e são mais leves e potentes que suas rivais de Li-Ion. Elas não sofrem com o efeito memória, mas possuem um custo muito elevado em função de seu custo de produção.

Nota: A tensão das baterias de Lítio não pode passar de um limite, elas devem variar apenas dentro de uma margem estabelecida pelo fabricante. Uma descarga excessiva pode comprometer a sua capacidade de armazenamento e uma carga excessiva pode causar até a sua explosão.

Nota: A simbologia usada para representar fontes de corrente continua está representada abaixo.



Aviso: Use sempre fontes próprias para o seu tipo de bateria com a tensão e corrente indicada pelo fabricante para evitar acidentes e prolongar a vida útil do produto.

4.3 Associações:

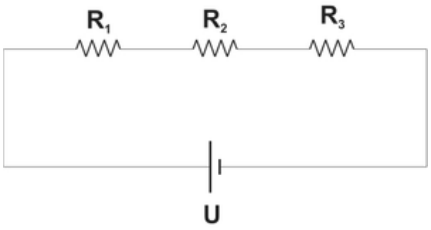
Muitas vezes se torna necessária a associação de componentes ou de pilhas para se obter o valor necessário para o projeto pelo fato de nem todos os valores estarem disponíveis comercialmente ou em sua disposição. As associações podem acontecer de três formas, em série, em paralelo ou mistas, e cada uma possui suas características.

- **Série:** A associação em serie ocorre quando um componente é colocado em seguida do outro.
- **Paralelo:** A associação em paralelo ocorre quando os componentes são colocados um ao lado do outro.
- **Mistas:** Consiste no uso de ambas as combinações anteriores ao mesmo tempo.

As características e as formas de associações serão melhores explicadas abaixo, quando cada componente for abordado.

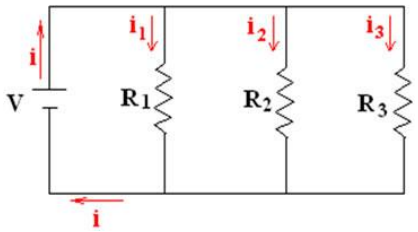
4.3.1 Resistores:

- Em série: os resistores são conectados um na frente do outro de forma que a corrente percorre todos. A resistência equivalente é igual à soma das n resistências em série.



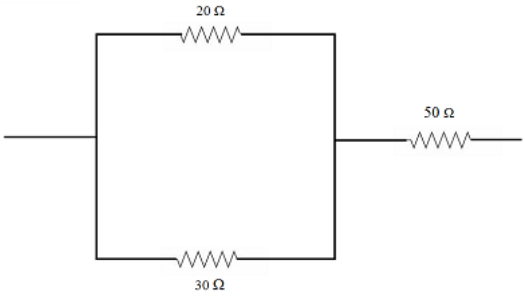
$$R_{eq} = R_1 + R_2 + \cdots + R_n$$

- Em paralelo: os resistores são conectados paralelamente, dessa forma, a corrente se divide entre eles. A resistência equivalente segue a fórmula a seguir:



$$\frac{1}{R_{eq}} = \frac{1}{R_1} + \frac{1}{R_2} + \cdots + \frac{1}{R_n}$$

- Mista: os resistores são conectados tanto em paralelo quanto em série. A resistência equivalente será calculada utilizando as fórmulas de associações em série e em paralelo.



4.3.2 Capacitores:

A associação de capacitores ocorre de maneira muito semelhante à de resistores, porém, deve-se usar a fórmula de associação em paralelo de resistores para a associação em série de capacitores e vice-versa, invertendo, assim, os casos.

4.3.3 Pilhas/Baterias:

Um conjunto de pilhas pode ser chamado de bateria e a sua associação deve ser feita com cautela. Ao associar, use apenas pilhas de mesma idade e de mesmo material e estude mais profundamente sobre como deve ser feito o carregamento que, muitas vezes, deve ocorrer de maneira individual para cada célula da bateria.

- **Série:**

  A associação em série de pilhas possibilita um aumento da tensão proveniente delas, ou seja, se o polo positivo de uma pilha é ligado ao negativo da seguinte a voltagem de ambas é somada.

- **Paralelo:**

  Por outro lado, a associação em paralelo não aumenta a tensão resultante, mas sim a corrente/duração das pilhas utilizadas.

- **Mista:**

  Como o próprio nome já sugere, esse tipo de associação utiliza ambas as técnicas para obter maior corrente e maior tensão. Suas grandezas de saída podem ser calculadas usando os mesmos meios dos itens anteriores.

5.Projetos com o Arduino

Nota: Todos os códigos dos projetos estão disponíveis em: http://github.com/RatosdePC/ApostilaBrino. Você pode baixa-los para evitar digitar todos os rascunhos que utilizaremos manualmente. Eles se encontram sob a pasta "Codigos", divididos por projeto.

5.1 Piscar:

Neste capítulo trabalharemos o mais simples dos projetos de robótica: piscar um LED. Mesmo que rudimentar, tal projeto é de grande importância para entender como funcionam os tipos de saída do Arduino e como utilizar a IDE do Brino.

Esse experimento pode ser feito tanto com o LED interno do Arduino, como também com um LED externo. Demonstraremos, inicialmente, como fazê-lo utilizando o interno.

O Código

Abra a IDE do Brino e digite o código a seguir:

```
// Projeto 1 – Piscar

Numero pinoLed = 13;

Configuracao(){
        Pino.definirModo(pinoLed, Saida);
}

Principal(){
        Pino.ligar(Digital.pinoLed);
        esperar(2000);
        Pino.desligar(Digital.pinoLed);
        esperar(2000);
}
```

Depois, conecte seu Arduino ao computador e verifique a porta serial que ele está conectado. Abra as ferramentas do IDE e selecione a porta e a placa que você está utilizando. Uma vez selecionado, clique em Verificar e Carregar. Se uma janela aparecer solicitando o nome do rascunho, insira "Piscar", clique em "OK" e espere até o log do IDE (área embaixo do editor) mostrar que o rascunho foi compilado e carregado. Caso ocorra qualquer erro, verifique o seu código, a placa e a porta serial selecionadas. Depois de carregado, observe o LED da sua placa acender por dois segundos e depois desligar também por dois segundos.

Analisando o código

A primeira linha do código é:

//Projeto 1 – Piscar

Essa linha é apenas um comentário com o nome do projeto.
A linha seguinte é:

Numero pinoLed = 13;

Essa linha cria uma variável do tipo (*Numero*) com o nome (*pinoLed)* para armazenar o número do pino em que conectamos o LED.
Logo depois, declaramos a função Configuracao():

```
Configuracao(){
        Pino.definirModo(pinoLed, Saida);
}
```

A função de configuração é um dos métodos obrigatórios a todo rascunho Brino e é responsável por preparar o necessário para a execução da função Principal(). Neste caso, ela possui apenas uma instrução em seu bloco de código:

```
Pino.definirModo(pinoLed, Saida);
```

Essa linha define o modo do *pinoLed*, que possui o valor 13, como (*Saida*), ou seja, o Arduino irá emitir uma corrente elétrica de 0 ou 5 V. O método *definirModo* do conjunto *Pino* tem como argumentos o número do pino e o modo, que pode ser *Entrada* ou, como neste caso, *Saida*. Depois de executar a configuração o Arduino inicia o método *Principal()* que nesse rascunho é:

```
Principal(){
        Pino.ligar(Digital.pinoLed);
        esperar(2000);
        Pino.desligar(Digital.pinoLed);
        esperar(2000);
}
```

A função (*Principal*) é o segundo método obrigatório a todo rascunho Brino e será repetido enquanto o Arduino estiver ligado. Sua primeira linha é:

```
Pino.ligar(Digital.pinoLed);
```

O método *ligar* do conjunto *Pino* liga o pino fornecido como argumento, no caso *Digital.pinoLed.* A indicação "*Digital.*" antes de nossa variável avisa para o Arduino que estamos utilizando um pino digital. Quando ligamos uma porta enviamos 5 volts para ela e quando desligamos enviamos 0 volts. A linha seguinte é:

```
esperar(2000);
```

A função esperar é um método Arduino que faz uma "pausa" na execução do código durante o número de milissegundos indicados entre os parênteses, no caso 2000, que equivale a 2 segundos. Depois de esperar o tempo definido o Arduino executa a próxima linha:

```
Pino.desligar(Digital.pinoLed);
```

Se a função *ligar* fazia com que o pino ligasse, o método *desligar* faz o contrário, desliga o pino. Ou seja, essa linha irá enviar 0V pelo pino digital *pinoLed*, fazendo o LED apagar. Depois disso o

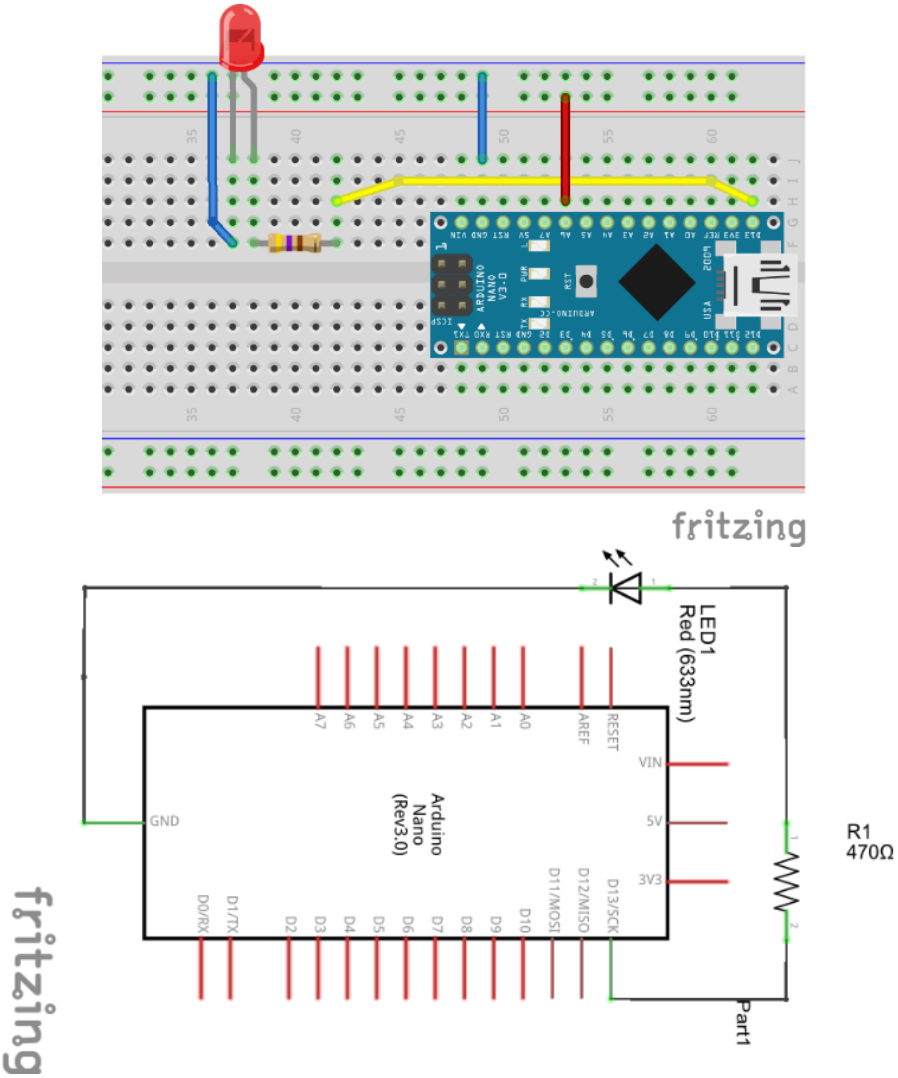
código apresenta outra espera de dois segundos e reinicia o método *Principal()*, repetindo-o enquanto o Arduino permanecer ligado.

Agora, antes de demonstrar esse projeto utilizando um LED externo, listamos os materiais necessários para a montagem do hardware, ou seja, a parte física do projeto:

- Protoboard
- LED
- Resistor de 470 ohms (amarelo, lilás, marrom)
- Jumpers

Montando o hardware:

Primeiramente, reproduza o circuito abaixo, com seu Arduino desligado. Se necessário consulte também o diagrama ou esquemático das ligações:



Analisando o hardware

O hardware montado para esse projeto é bem simples. Conectamos à porta 13 do Arduino um resistor de 470 Ω e, ao resistor, um LED que tem sua outra perna conectada ao ground do circuito. O resistor é utilizado para provocar uma queda de corrente e tensão no circuito, pois o Arduino envia, em cada porta, quando ligada, 5V de tensão e 40 mA de corrente e seu LED, geralmente utiliza uma corrente de 20 mA e uma tensão abaixo dos 3V. Para calcular o melhor resistor para seu circuito, deve-se buscar o datasheet do LED e verificar a corrente e tensão nominais dele. Com esses dados em mãos, aplica-se a seguinte fórmula:

$$R = \frac{(V_s - V_l)}{i}$$

Em que R é o valor da resistência que deve ser usada; V_s é a voltagem fornecida (no caso os 5V do Arduino); V_l é a voltagem nominal do LED; e i é a corrente nominal do LED.

Nota: O datasheet é um documento que contém todas as informações técnicas de um determinado componente

5.2 Ligar luz com botão

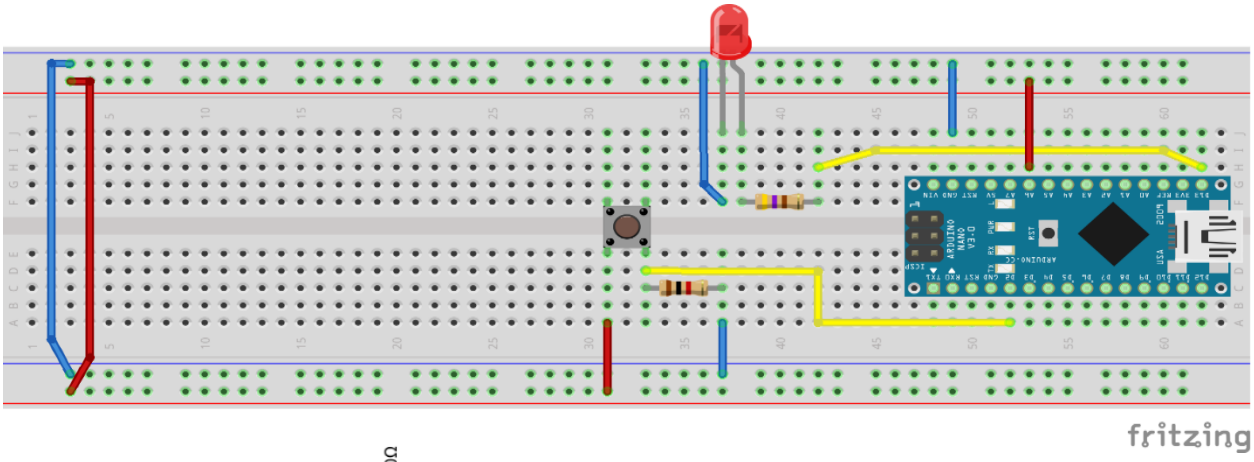
Neste projeto vamos utilizar um botão para controlar o LED do último projeto. Durante o desenvolvimento desse projeto vamos aprender como utilizar entradas digitais no Arduino com resistores *pull-down* externos e *pull-up* internos. As entradas digitais são aquelas que possuem apenas dois valores, Ligada ou Desligada. Para desenvolver esse projeto você precisará de:

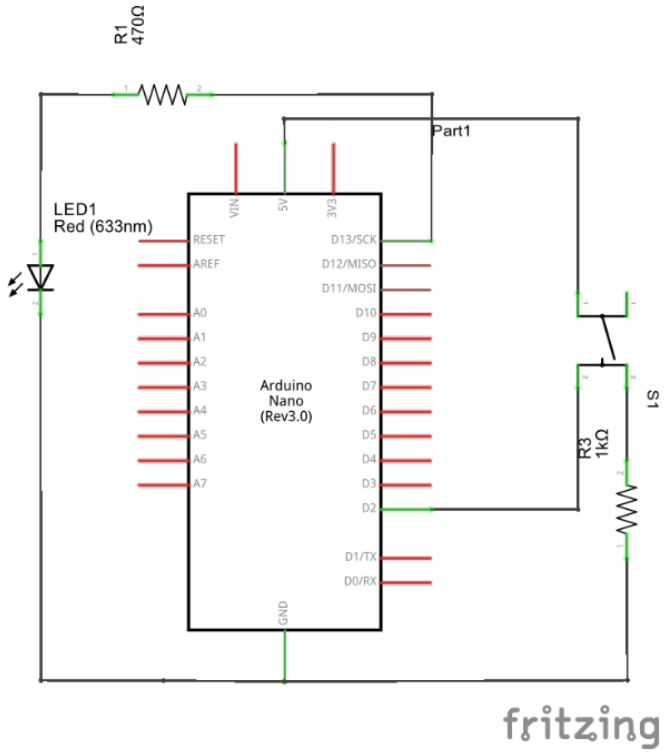
- Protoboard
- LED
- Resistor de 470 ohms
- Resistor de 1K ohm
- Botão (Interruptor tátil)

Montando o hardware

Com o seu Arduino desconectado, monte o circuito mostrado abaixo:

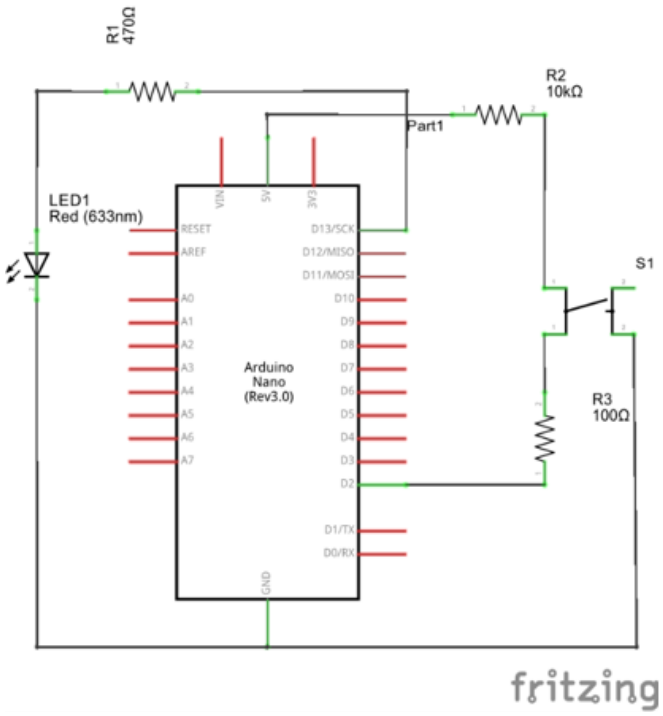
Nota: nunca monte seus circuitos com o seu Arduino conectado/ligado.





Analisando o hardware

O hardware desse projeto é muito parecido com o do primeiro, com a adição do botão na porta digital 2 do Arduino. Para montá-lo, utilizamos um resistor *pull-down* de 1k ohm. Os resistores *pull-down* e *pull-up* são utilizados como filtros para as portas digitais do Arduino, para garantir que elas não leiam valores aleatórios. A diferença entre eles está no estado que eles mantêm na porta do Arduino. Resistores *pull-down* mantém valores baixos ou desligados enquanto o botão não for pressionado, pois conectam o ground (0V) à porta digital. Já os resistores *pull-up* fariam uma inversão dos estados: manteriam a porta alta, ou ligada (5V), enquanto o botão não estivesse pressionado. Abaixo mostramos uma imagem da montagem de resistores *pull-up* no circuito do projeto 2.



O Código

Abra a IDE do Brino e digite o código abaixo ou baixe-o da página do Github da apostila.

```
//Projeto 2 – Ligar luz com botão

Constante Numero pinoLed = 13;
Constante Numero  botao = 2;

Configuracao(){
        Pino.definirModo(pinoLed, Saida);
        Pino.definirModo(botao, Entrada);
}

Principal(){
        Numero estadoBotao = Pino.ler(Digital.botao);
        se (estadoBotao == Ligado){
                Pino.escrever(Digital.pinoLed, Ligado);
                esperar(1000);
        }
        Pino.escrever(Digital.pinoLed, Desligado);
}
```

Analisando o código

A partir desse capítulo destacaremos apenas métodos e palavras-chaves não descritas anteriormente.

Constante Numero pinoLed = 13;

Diferente do outro código, dessa vez marcamos a variável *pinoLed* como uma constante, ou seja, avisamos para o Arduino que seu valor não será alterado durante a execução do rascunho.

Pino.definirModo(botão, Entrada);

Nessa linha definimos que o nosso botão atuará como uma entrada e não como uma saída, dessa forma o Arduino poderá ler o valor da porta e saber se o botão está apertado ou não.

Numero estadoBotao = Pino.ler(Digital.botao);

A partir do método *ler* do conjunto *Pino* podemos obter o valor da porta digital *botao* que poderá ser *Ligado*, caso exista corrente fluindo, ou *Desligado*, caso não exista corrente fluindo. Depois de ler a porta, guardamos o valor na variável *estadoBotao*.

```
se (estadoBotao == Ligado){
        ...
}
```

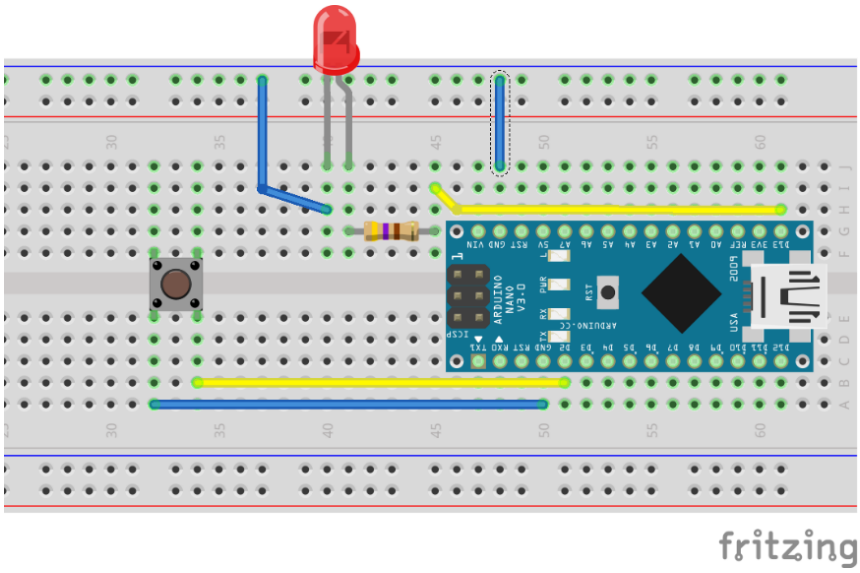
Depois de ler a porta, criamos um bloco condicional que será executado *se* a porta estiver ligada, ou seja, se o botão estiver apertado. Caso ele esteja solto, o Arduino simplesmente ignorará todo o código dentro do bloco do *se*.

Pino.escrever(Digital.pinoLed, Ligado);

O método *escrever* nada mais é do que outra forma de ligar e desligar o pino. No primeiro projeto utilizamos os métodos *ligar* e *desligar*, que equivalem a escrever o estado *Ligado* ou *Desligado*, respectivamente, ao pino. Os métodos *ligar* e *desligar* são abstrações, ou seja, simplificações, do método *escrever*.

Pull-up

O Arduino já possui internamente resistores pull-up. É possível utilizar o modo *Entrada_PULLUP* para aproveitar estes resistores. A montagem e o código ficariam da seguinte forma:




```
//Projeto 2.1 – Ligar luz com botão pull-up

Constante Numero pinoLed = 13;
Constante Numero  botao = 2;

Configuracao(){
        Pino.definirModo(pinoLed, Saida);
        Pino.definirModo(botao, Entrada_PULLUP);
}

Principal(){
        Numero estadoBotao = Pino.ler(Digital.botao);
        se (estadoBotao == Desligado){
                Pino.escrever(Digital.pinoLed, Ligado);
                esperar(1000);
        }
        Pino.escrever(Digital.pinoLed, Desligado);
}
```

Como você pode perceber a leitura do botão fica invertida, logo, quando ele é pressionado, a leitura será igual a 0V ou desligado.

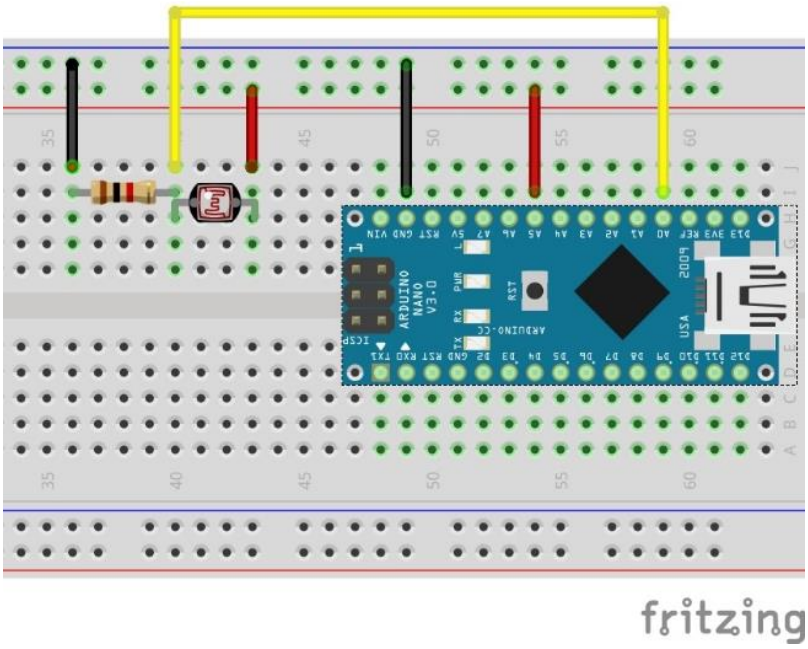
5.3 Leitura Analógica para USB

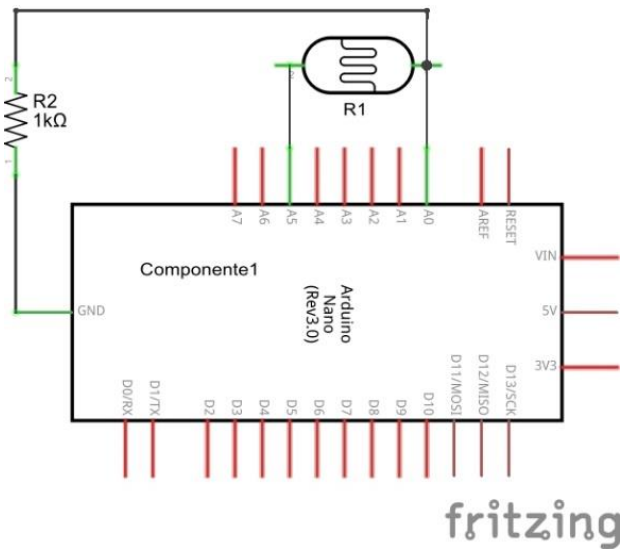
Nesta unidade aprenderemos a fazer leituras de um LDR, um sensor resistivo muito utilizado para medir diferenças de luminosidade. Esse será o segundo sensor abordado nessa apostila, mas o primeiro que utiliza entradas analógicas. Durante o desenvolvimento desse projeto, aprenderemos a utilizar a porta analógica do Arduino. Para desenvolvê-lo você precisará de:

- Protoboard
- LDR (Light Dependant Resistor)
- Resistor de 1K ohm; (Marrom, Preto, Vermelho)

Montando o hardware

Com o Arduino desconectado, monte o circuito mostrado abaixo:





Analisando o hardware

O hardware desse projeto é extremamente simples. O Arduino alimenta o circuito por meio da porta 5V, a corrente flui pelo LDR, que de acordo com a luz do ambiente varia sua resistência, alterando a quantidade de energia que passa por ele. Em seguida, essa energia volta para a placa, fechando o circuito. Há um resistor conectado ao GND do Arduino e a uma das extremidades do LDR (esse componente não possui polaridade), atuando como um filtro e estabilizando a tensão de saídas. Essa mesma perna do LDR ligada ao resistor é conectada na porta A0 do Arduino, para que possamos fazer as leituras necessárias.

O Código

Abra a IDE do Brino e digite o código abaixo ou baixe-o da página do GitHub da apostila.

```
//Projeto 3 – Leitura Analógica Para USB

Numero Constante ALDR = 0;
Numero leitura;

Configuracao() {
        USB.conectar(9600);
}

Principal() {
        leitura = Pino.ler(ALDR);
        USB.enviarln(leitura);
        esperar(500);
}
```

Analisando o código

O código começa definindo a porta de entrada do sensor e, em seguida, cria a variável que vai guardar o valor das medidas. Essa parte do código está exposta abaixo:

```
Numero LDR = 0;
Numero leitura = 0;
```

Depois, inicia-se a comunicação serial via USB para que seja possível apresentar os dados lidos pelo Arduino na tela do computador:

```
USB.conectar(9600);
```

A primeira linha do Principal() é:

```
leitura = Pino.ler(ALDR)
```

Ela faz a leitura da porta analógica onde está conectado o LDR (no caso a porta A0) e atribui o valor dessa medida a variável "*leitura*" para que possa ser exibida tela pela próxima linha.

```
USB.enviarln(leitura);
```

Por fim, o código possui uma pausa de meio segundo entre uma medida e outra, para facilitar o controle dos dados obtidos.

```
esperar(500);
```

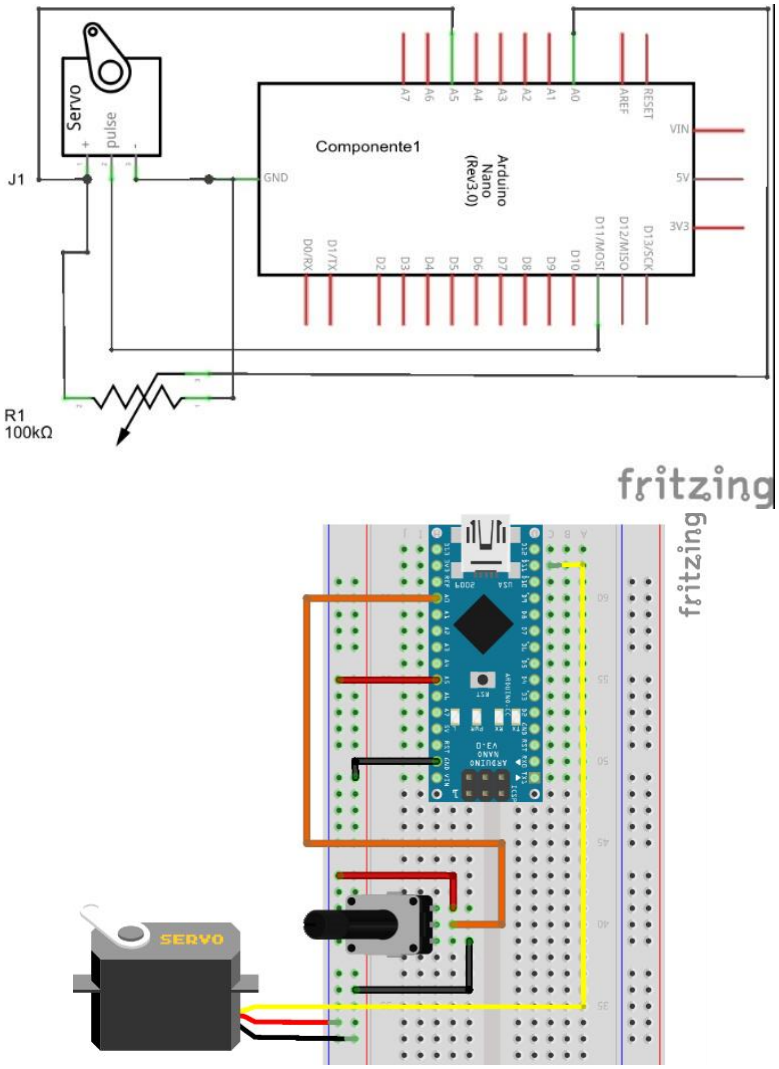
5.4 Servo controlado por potenciômetro

Neste capítulo aprenderemos como controlar um servo motor utilizando um Arduino com dados recebidos de um potenciômetro. Para isso teremos que lidar com portas analógicas para realizar a comunicação do Arduino com o potenciômetro e com o servo motor. O material necessário para a montagem desse circuito é:

- Sevo motor
- Protoboard
- Potenciômetro de 10K ohms

Montando o Hardware

Com o Arduino desconectado, monte o circuito abaixo:



Analisando o Hardware

O hardware desse projeto é dividido em duas partes, a do o servo motor e o a do potenciômetro. O servo motor é conectado na alimentação, que nesse caso vem do próprio Arduino, e o seu sinal é controlado por uma porta digital. Já o potenciômetro deve ter seus pinos mais externos conectados ao 5V e ao GND e seu pino do meio, por onde sai o nosso sinal, deve ser lido por uma porta analógica. Para isso, usaremos a porta A0 do Arduino.

O Código

```
//Projeto 4 – Servo Controlado por Potenciômetro

usar Servo
Servo meuServo;

Numero Constante Apotenciometro = 0;
Numero valorPotenciometro;
Numero angulo;

Configuracao() {
        meuServo.conectar(Digital.5);
}

Principal(){
        valorPotenciometro = Pino.ler(Apotenciometro);
        angulo = proporcionar(valorPotenciometro, 0, 1023, 0, 180);
        meuServo.escreverAngulo(angulo);
        esperar(15);
}
```

Analisando o código

O código desse projeto começa com a adição da biblioteca para controle de servos e, em seguida, nós damos um nome ao servo:

usar Servo
Servo meuServo;

Depois disso, as variáveis que iremos usar são declaradas e o bloco de Configuracao() possui apenas uma linha que é:

meuServo.conectar(Digital.5);

Essa linha está associando o servo (*meuServo*) a porta 5 usando o comando conectar. No loop *Principal()* podemos observar que ocorre a leitura do nosso potenciômetro (*Apotenciometro*) que salva na nossa variável (*valorPotenciometro*). Em seguida ele faz uma regra de três entre a entrada, que varia de 0 a 1023, com a saída digital para o servo e salva esse valor na variável *angulo*.

angulo = proporcionar(valorPotenciometro, 0, 1023, 0, 180);

Em seguida é feito o ajuste do servo motor com base na variável enviada usando o comando *escreverAngulo*.

meuServo.escreverAngulo (angulo);

5.5 Ultrassom + Memória

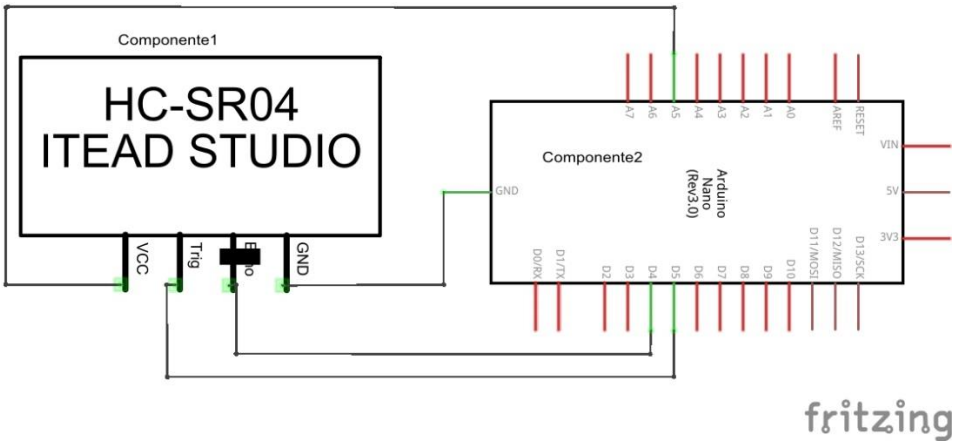
Neste capítulo, vamos montar um medidor de distâncias que registrará os valores obtidos na memória interna do Arduino, mostrando-os pela porta USB depois. Este será o primeiro projeto que utilizará uma biblioteca externa do Arduino. Caso você não saiba instalá-las, existe um capítulo no final do livro explicando como fazer isso. Ela se chama Ultra e está disponível em "*github.com/RatosDePC/Ultra*".

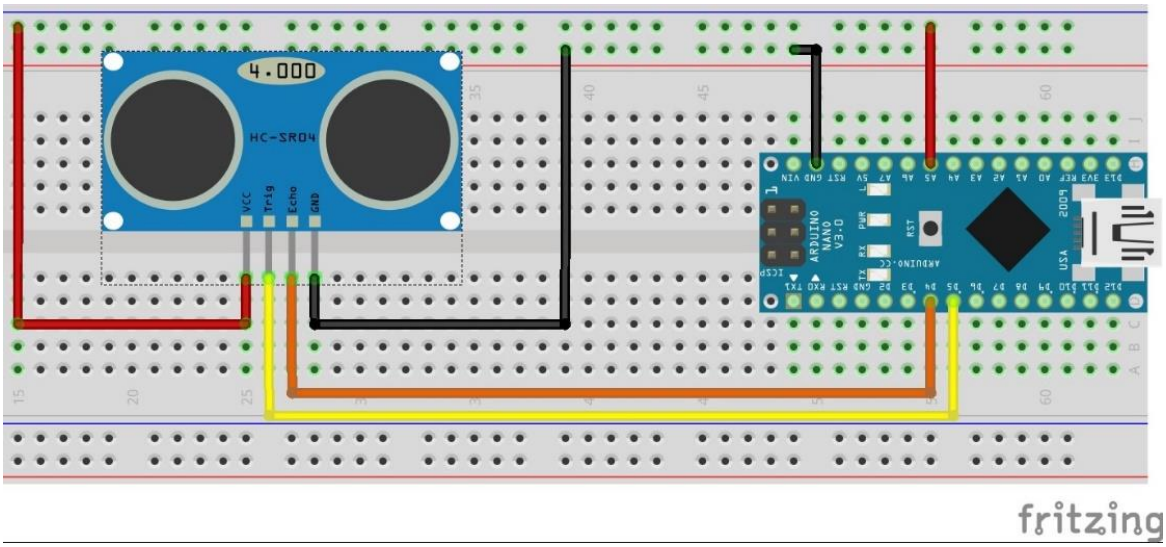
O material necessário para essa prática é:

- Protoboard
- Sensor ultrassônico HC- sr04

Montando o Hardware

Com o Arduino desconectado, monte o circuito abaixo:





Analisando o Hardware

O hardware desse projeto é bem simples. A porta 5 é conectada ao *trigger* do ultrassônico, identificado no sensor como TRG, que é a porta de controle. Para ativar o envio de pulsos, manteremos essa porta ligada por 10 us. E o pino 4 será ligado ao *echo*, identificada como ECH, que transmite a resposta por meio da duração do pulso para o arduino. Além das linhas de dados, conectam-se os pinos de alimentação do sensor aos respectivos da placa microcontroladora – 5V ao 5V e GND ao GND.

O Código

Abra a IDE do Brino e digite o código a seguir:

```
//Projeto 5 – Ultrassom + Memoria

usar Ultra
usar Memoria

Ultra u(5,4);

Configuracao(){
	USB.conectar(9600);
	para( Numero x = 0; x < 5; x++){
		Numero d = u.medir();
		Memoria.escrever(x, d);
		USB.enviarln(d);
	}
	para( Numero x = 0; x < 5; x++){
		USB.enviarln(Memoria.ler(x));
	}

}

Principal(){

}
```

Analisando o código

A primeira linha do código é:

usar Ultra

Essa linha irá importar a biblioteca *Ultra* para podermos utilizar seus métodos durante a execução, ou seja, avisamos ao Arduino que precisaremos deles para fazer o que queremos. A próxima linha também é uma importação:

usar Memoria

Nessa linha, avisamos o Arduino que utilizaremos a memória disponível internamente e os métodos associados à sua utilização.
Logo depois temos a linha:

Ultra u(5,4);

Essa linha cria um objeto *Ultra* que tem sua porta trigger (que envia o pulso ultrassônico) ligada ao pino 5, e a porta eccho (que notifica a recepção do pulso) ao pino 4. O método *Configuracao* () começa inicializando a conexão *USB* para podermos verificar o bom funcionamento do nosso código. Depois temos um loop *para*. Esse loop cria uma variável que utilizaremos de contador e enquanto ela estiver no parâmetro definido, no caso, menor que 5, ele repetirá seu bloco realizando o incremento definido ao final, em nosso código, *x++*.

para(Numero x; x < 5; x++){

O computador repetirá o código 5 vezes (pois após esse número de repetições, x será maior ou igual a 5). O bloco de código do *para* começa com a seguinte linha:

Numero d = u.medir();

Essa linha cria uma variável pra guardar o valor da distância medida pelo ultrassônico com o método *medir()*. Depois temos:

Memoria.escrever(x, d);

Essa linha irá guardar a distância *d* no endereço *x* da memória, onde *x* é o contador do loop *para*. Além de guardar na memória iremos enviar para a porta USB para podermos verificar o valor lido pelo Arduino. Em seguida, repetiremos o *para* com um código que, ao invés de escrever, irá ler o que gravamos na memória e mostrar embaixo do que o primeiro *para* mostrou. A linha que faz isso é:

USB.enviarln(Memoria.ler(x));

O nosso método *Principal* está vazio, pois não queremos que o Arduino repita todo esse processo, só queremos que ele seja executado uma vez.

5.6 Carrinho com servo de rotação contínua

Nesse projeto, aprenderemos como dar comandos básicos para um par de servos motores de rotação contínua, ou seja, como controlar uma base simples. Caso você não tenha uma à disposição, pode-se simplesmente improvisar uma, como também mostraremos nesse capítulo. Caso já possua uma base, basta acoplar o Arduino e os motores a ela para começar a se divertir! Para esse projeto serão necessários os seguintes materiais:

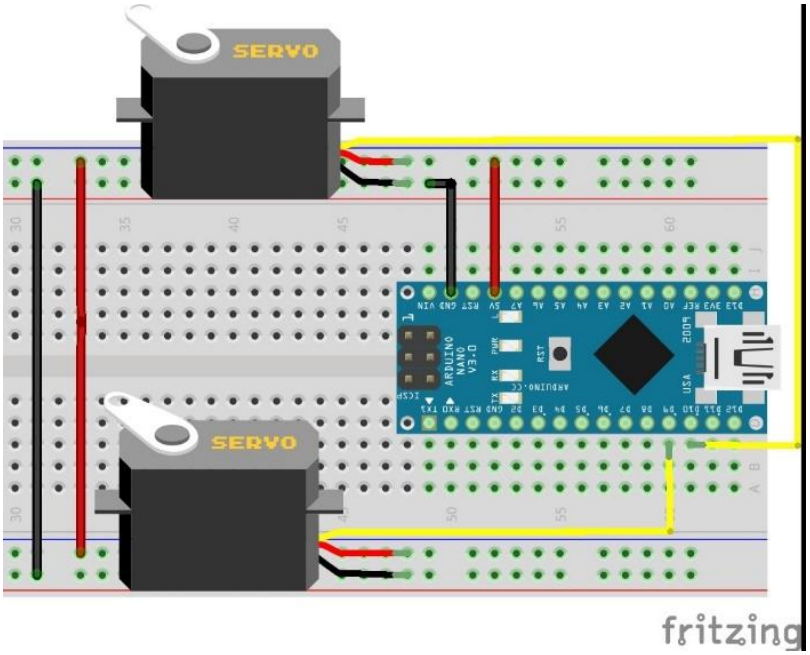
- 2 servos motores de rotação contínua
- Base com rodas ou esteiras (opcional)
- Protoboard
- Alimentação

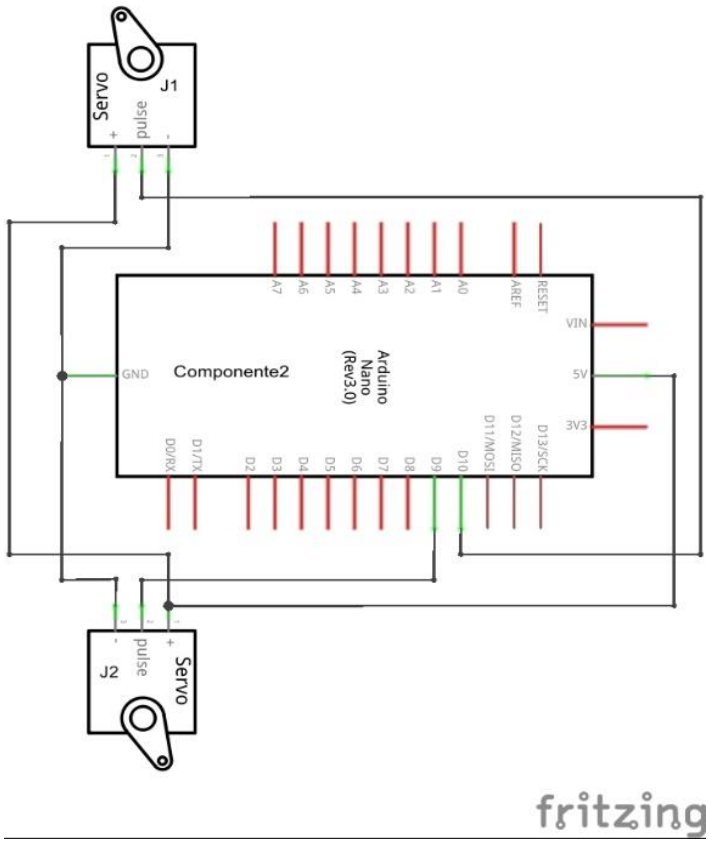
Base caseira

Não se preocupe, na falta de uma base, basta unir os dois servos utilizando fita adesiva e colocar algum suporte sobre eles, para suportar os componentes. Praticamente qualquer coisa pode funcionar como uma base – palitos de churrasco e chapas de metal são boas opções- então seja criativo! O que importa é que seu suporte seja eficiente.

Montando o Hardware

Com o Arduino desconectado, monte o circuito abaixo:





Analisando o Hardware

Neste hardware temos dois servos conectados às portas 5V e GND do Arduino, com seus sinais (fio amarelo da imagem acima) nas portas D9 e D10. Conecte tudo a uma fonte (pilhas ou baterias) para que o carrinho tenha mais mobilidade. Lembre-se que é importante regular os servos de rotação contínua para que eles parem quando receberam a instrução certa. Para realizar a calibragem, escreva um rascunho que roda o comando *escreverMicros* para ambos os motores utilizando como parâmetro a constante *Servo.parar*.

O Código

Abra a IDE do Brino e digite o código.

```
//Projeto 6 – Carrinho com servo de rotação contínua

usar Servo

Servo servoD;
Servo servoE;

Configuracao(){
	servoD.conectar(Digital.9);
	servoE.conectar(Digital.10);
}

Principal(){
	Numero x = 0;
	enquanto (x<3){
		servoD.escreverMicros(Servo.frente);
		servoE.escreverMicros(Servo.frente);
		esperar(2000);
		servoD.escreverMicros(Servo.frente);
		servoE.escreverMicros(Servo.tras);
		esperar(2000);
		x++;
	}
	servoD.escreverMicros(Servo.tras);
	servoE.escreverMicros(Servo.frente);
	esperar(2000);
	servoD.escreverMicros(Servo.tras);
	servoE.escreverMicros(Servo.tras);
	esperar(2000);
}
```

Analisando o código

O código começa com a adição da biblioteca 'Servo', que é seguida pela escolha do nome dos servos, *servoD* para o da direita e *servoE* para o da esquerda. No loop *Principal()* é declarada uma

variável, nomeada *x*, que será usada como um contador. Em seguida o robô deve executar uma série de movimentos expressos por linhas como:

```
servoD.escreverMicros(Servo.frente);
servoE.escreverMicros(Servo.tras);
```

Cada movimento é executado por 2 segundos e, no final do loop *enquanto()* a variável *x* é acrescida em uma unidade até que esse loop acabe, ou seja, até *x* não ser mais menor do que 3.

```
x++;
```

5.7 Robô Ultrassônico

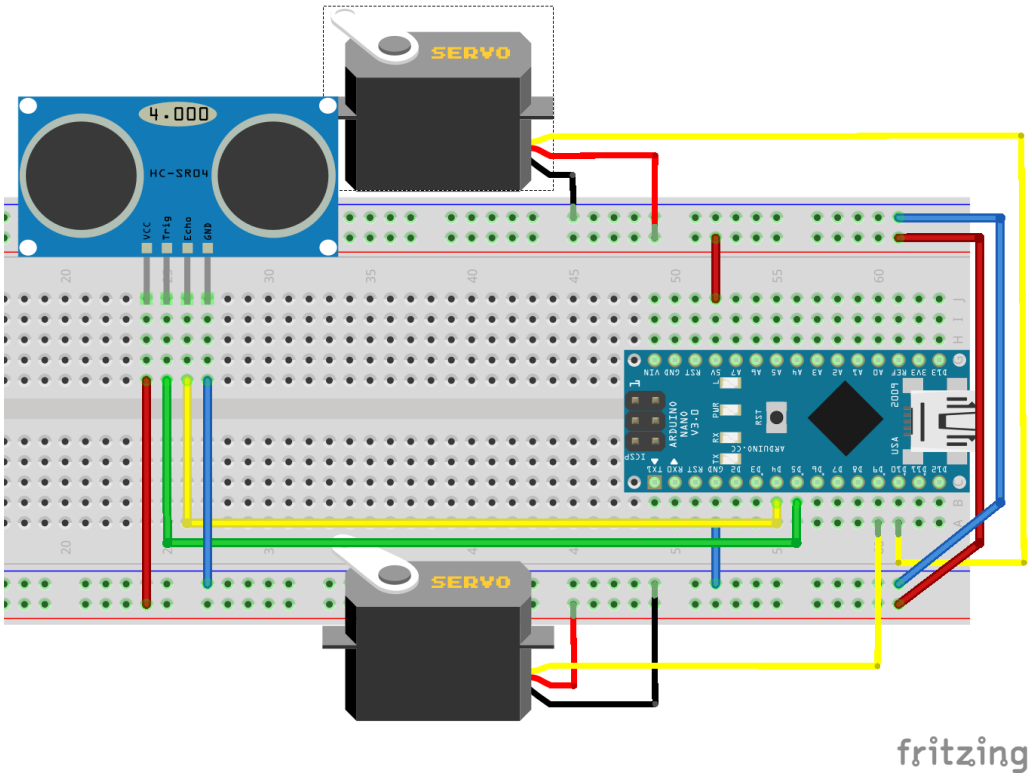
Neste capítulo, utilizaremos os conteúdos ensinados, principalmente nos capítulos do sensor ultrassônico e do carrinho com servo de rotação contínua, para montar um projeto um pouco maior. Este robô utilizará um sensor ultrassônico para desviar de obstáculos. Quando ele detectar algum objeto a menos de centímetros, ele irá virar para um dos lados e depois continuará a andar para a frente. Neste projeto, precisaremos da mesma base utilizada na montagem anterior, ou algo parecido. Os materiais necessários estão descritos abaixo:

- Protoboard
- Sensor Ultrassônico HC-sr04
- 2 servos de rotação contínua
- Base de montagem
- Rodas
- Jumpers

Depois de juntar o material necessário, vamos começar a montagem.

Montando o hardware

O hardware desse projeto é composto pela união dos projetos do ultrassônico e do carrinho, como mencionado anteriormente. A parte elétrica deve ser montada de acordo com as imagens e o esquemático abaixo:



Analisando o hardware

O hardware desse projeto não apresenta nenhuma inovação em relação aos projetos já montados. O ultrassônico utilizará as mesmas portas utilizadas anteriormente e os servos também. Vale ressaltar que o Arduino possui uma limitação de corrente, portanto, se fossem utilizados mais servos ou sensores, o ideal é que se utilize uma fonte externa de alimentação.

Nota: Lembre-se de realizar a calibração dos motores antes de colocar o seu robô para funcionar.

O Código

Abra a IDE do Brino e copie o seguinte código:

```
// Projeto 7 – RoboUltrassonico

usar Servo
usar Ultra

Ultra u(5,4);
Servo servoD;
Servo servoE;

Configuracao(){
        servoD.conectar(Digital.9);
        servoE.conectar(Digital.10);
}

Principal(){
        enquanto (u.medir() > 5){
                //caso seus motores rotacionem para o lado errado, troque Servo.frente por
                //Servo.tras
                servoD.escreverMicros(Servo.frente);
                servoE.escreverMicros(Servo.frente);
        }
        servoD.escreverMicros(Servo.frente);
        servoE.escreverMicros(Servo.tras);
        //o tempo de espera para uma curva de 90 graus pode variar
        esperar(3000);
}
```

Analisando o código

O código deste projeto começa importando as bibliotecas que precisaremos para controlar o servo e o sensor ultrassônico.

usar Servo
usar Ultra

Depois de importá-las, vamos criar os objetos correspondentes aos seus objetos, que representarão os servos da esquerda e da direita e o sensor ultrassônico.

```
Ultra u(5,4);
Servo servoD;
Servo servoE;
```

Esses serão os objetos que usaremos de referência para controlar as partes do robô durante a execução do código. Depois disso, abrimos o método *Configuracao*() que possui as seguintes linhas:

```
servoD.conectar(Digital.9);
servoE.conectar(Digital.10);
```

Essas linhas vão definir as portas de sinal dos servos de rotação contínua. Ele define a porta D9 para o motor direito e a porta D10 para o esquerdo. Isso é tudo que precisamos configurar para nosso rascunho funcionar. Vamos partir então para o método *Principal()*, cuja primeira linha é a abertura de um loop *enquanto*:

```
enquanto(u.medir > 5){
        ...
}
```

Esse loop irá repetir seu bloco enquanto o sensor ultrassônico não medir distâncias menores que 5 centímetros. Ou seja, enquanto não houverem obstáculos na frente do robô. Dentro do bloco temos dois comandos equivalentes, apenas para motores diferentes.

```
servoD.escreverMicros(Servo.frente);
servoE.escreverMicros(Servo.frente);
```

Esses linhas enviam pulsos para os motores para que ambos rodem para o mesmo lado, utilizando a constante *Servo.frente*, cujo valor é 1700. Ou seja, seu robô continuará em frente até que encontre algo. Caso seus motores estejam rodando um para cada lado, troque a constante do que estiver rodando para trás por *Servo.tras*, cujo valor é 1300. Lembre-se de alterar também o valor da constante deste motor fora do loop *enquanto*, caso necessário. Fora do loop, temos duas linhas que invertem a rotação de um dos motores, para que o robô vire e uma chamada ao método *esperar*, para deixarmos que esse complete uma curva de 90°, aproximadamente.

```
servoD.escreverMicros(Servo.frente);
servoE.escreverMicros(Servo.tras);
esperar(3000);
```

Podem ser necessários ajustes no tempo de espera de acordo com a velocidade do seu motor, diâmetro da roda ou comprimento da esteira. Caso seu robô esteja virando menos que 90°, aumente o tempo de espera, caso esteja virando mais, diminua.

6. Despedida

Esse foi o último projeto de nosso livro. Esperamos que tenha sido possível ensinar-lhe o básico da linguagem Brino, que desenvolvemos, e da eletrônica e que você não pare por aqui. O potencial de crescimento do mercado da engenharia mecatrônica, elétrica, computação e várias outras que tiram proveito das habilidades descritas neste livro é enorme. Esse material não é capaz de te tornar um especialista, mas acreditamos que pode despertar a paixão necessária para tal, além de fornecer conceitos importantes.

O desenvolvimento do Brino e deste material tem como principal motivação a difusão da robótica nas escolas. Acreditamos que a robótica educacional tem um poder transformador muito grande. Por meio dela é possível despertar o gosto pela ciência e pela tecnologia em estudantes do ensino fundamental e médio, transformando jovens em potenciais pesquisadores. Não apenas isto, a robótica também auxilia o desenvolvimento de diversas habilidades importantíssimas no mercado de trabalho do mundo globalizado, como por exemplo, programação, solução de problemas complexos e gestão de equipes.

Além do desenvolvimento de habilidades específicas, a robótica pode auxiliar também no ensino de física em diversos níveis, em especial relacionando-se à eletrodinâmica, lecionada ao terceiro ano na grade curricular do Distrito Federal e no ensino médio no Brasil como um todo. Não apenas na área de física é possível se tirar proveito da robótica no ensino, a matemática também utiliza muito da lógica de programação, além da química e outras matérias no campo das exatas em geral.

Tendo em vista os enormes benefícios do ensino da robótica, além da diversão proporcionada por ela, nós cinco nos unimos para desenvolver o Brino e esta apostila para contribuir com a difusão do seu ensino nas escolas pelo país de forma mais acessível. Esse material é todo disponibilizado na internet, assim como o Brino, pois nós buscamos a democratização da robótica para causar o maior impacto positivo possível. Além disso, acreditamos na filosofia OpenSource.

Por fim, desejamos boa sorte para você em seus futuros projetos com o Arduino. Compartilhe seus projetos conosco e com a comunidade Brino em facebook.com/BrinoIDE. E conte com o nosso apoio para o desenvolvimento deles!

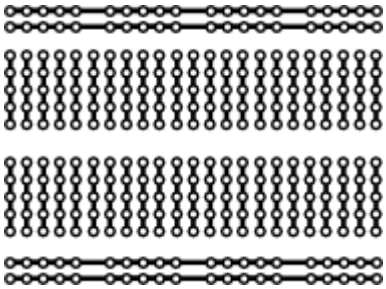
Equipe Brino

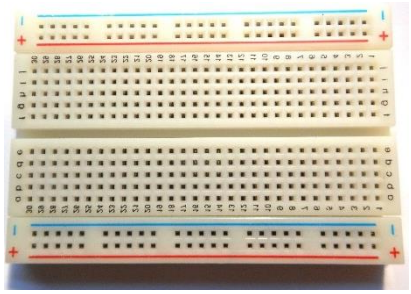
7.Apêndices

7.1 Materiais Importantes

7.1.1 Protoboard:

Também conhecida como breadboard, é uma maneira simples e eficaz de se realizar simulações de circuitos sem o uso de soldas, altamente recomendada para prototipagem. Caracteriza-se como uma placa de plástico repleta de entradas, ligados por uma malha metálica como está indicado no diagrama abaixo:





7.1.2 Jumpers:

Jumpers são fios usados para fazer ligações, como as realizadas na protoboard. Caracterizados como fios convencionais com segmentos de fios rígidos em suas extremidades para facilitar o seu uso para prototipagem. Eles podem ser facilmente conectados aos furos da protoboard ou de alguns Arduinos. Segmentos de fio soldados a uma placa de circuito impresso com o objetivo de ligar dois pontos desta também podem ser chamados de jumpers.

Nota: Eles geralmente representam as linhas que fazem as ligações em esquemáticos!!!

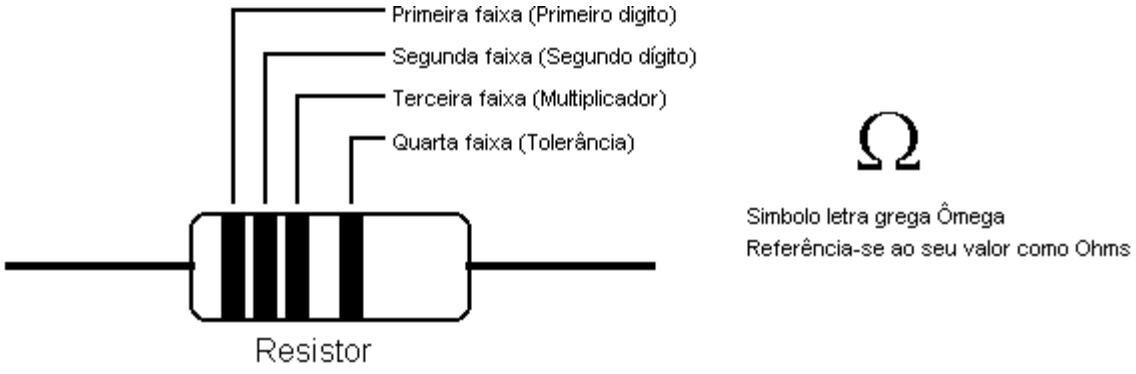
7.1.3 Fonte de alimentação:

É um aparelho capaz de gerar corrente contínua (CC) a partir da corrente alternada (AC) de uma tomada. É possível regular a voltagem e a corrente a partir de potenciômetros normalmente localizados na parte dianteira do aparelho.

7.2 Tabelas importantes

7.2.1 Valor de resistores:

A unidade de medida de resistência é o ohm, representado pelo símbolo grego Ômega. Para saber a resistência de um resistor basta ler as duas primeiras faixas e multiplicar esse valor pela terceira. A quarta mostra a tolerância do resistor, que é o valor de sua variação em relação ao valor nominal. Alguns resistores possuem 5 cores, sendo a quarta o multiplicador e a quinta a tolerância.



Cor	Valor	Multiplicador	Cor quarta faixa	Tolerância
Dourado	-	0.1	Prata	10%
Preto	0	1	Ouro	5%
Marrom	1	10	Amarelo	4%
Vermelho	2	100	Laranja	3%
Laranja	3	1.000	Vermelho	2%
Amarelo	4	10.000	Marrom	1%
Verde	5	100.000	-	-
Azul	6	1000.000	-	-
Violeta	7	Não existe	-	-
Cinza	8	Não existe	-	-
Branco	9	Não existe	-	-

7.2.2 ASCII:

A memória do computador não é capaz de armazenar diretamente caracteres, tendo que os armazenar na forma de números. Cada caractere possui, por conseguinte, o seu equivalente em código numérico: é o **código ASCII** (*American Standard Code for Information Interchange* - Código Americano Padrão para a Troca de Informações). Existem versões estendidas desse código, mas aqui trataremos da sua versão básica que possui 7 bits, ou seja, possui 128 caracteres.

Nessa tabela o código 0 a 31 não são realmente caracteres, sendo chamados de *caracteres de controle*. Os códigos 65 a 90 representam as letras maiúsculas e os códigos 97 a 122 representam as letras minúsculas. Abaixo representamos a tabela a partir do código 32.

Nota: Bastar somar ou subtrair 32 ao código ASCII para trocar entre as letras maiúsculas e minúsculas. Isso representa a troca do 6° bit da representação binária.

Código	Caractere	Código	Caractere	Código	Caractere
32	ESPAÇO	64	@	96	`
33	!	65	A	97	a
34	“	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	‘	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	underline	127	DEL

7.2.3Leds:

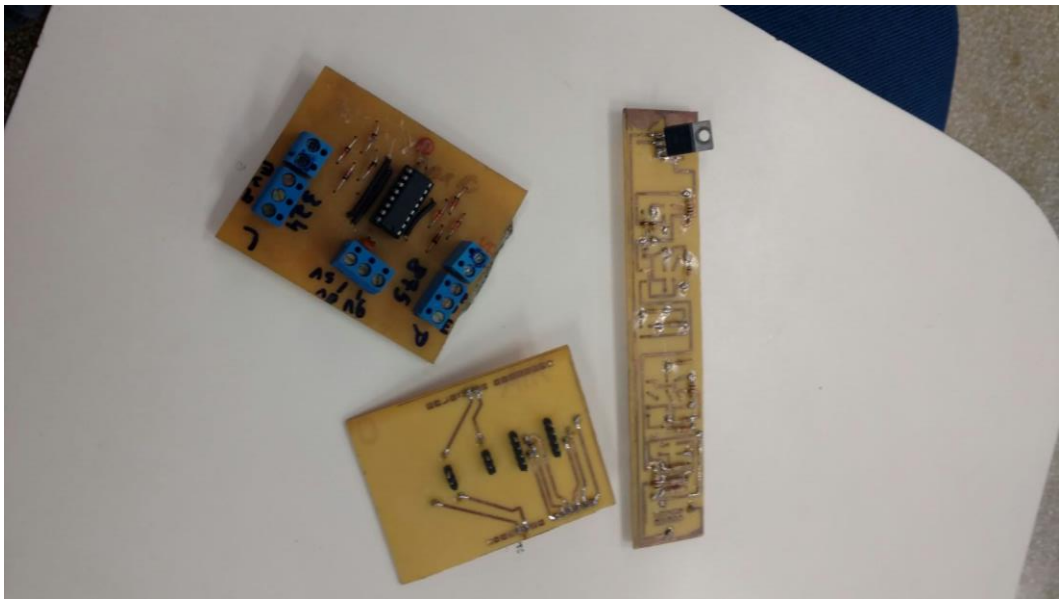
Para que cada LED possa ser utilizado, ele precisa de uma tensão e de uma corrente específica. Para saber como escolher as medidas certas basta saber o tipo de LED com que se está trabalhando e seguir a tabela abaixo:

LED	Tensão (V)	Corrente (mA)	LED	Tensão (V)	Corrente (mA)
Infravermelho 940 nm	1,5	50	Vermelho super brilhante	1.85	20
Vermelho normal	1,7	10	Amarelo normal	2,1	10
Infravermelho 880 nm	1,7	50	Laranja normal	2,1	10
Azul brilhante	3,6	20	Verde normal	2,2	20

7.3 Habilidades especiais

7.3.1 Placas de circuito impresso

Neste capítulo, ensinaremos como confeccionar placas de circuito impresso. Se você não sabe o que são PCI's (Placas de Circuito Impresso), elas são as placas dos equipamentos eletrônicos onde ficam soldados os componentes. Essas placas possuem trilhas de cobre que fazem as ligações do circuito. Abaixo é possível ver algumas das placas que fizemos para nossos projetos pessoais:



O processo de confecção começa com o desenvolvimento do esquemático das ligações e o teste do circuito em uma protoboard. Depois de montado e testado, deve-se fazer uma versão digital do circuito em um programa específico para isso. Existem vários no mercado, entretanto recomendamos que utilizem o *Eagle* da CAD Soft, por possuir uma versão grátis e por ser um dos melhores disponíveis. Outro programa bom, porém pago, é o Proteus. Para instalar o Eagle acesse: *http://www.cadsoftusa.com/download-eagle/freeware/*.

Depois de instalar o *Eagle*, você pode buscar vídeos ou outros tutoriais sobre sua utilização. Depois de montar o esquemático e o design da placa de circuito impresso com todas as trilhas necessárias, vamos exportar o desenho das trilhas utilizando o *Eagle*. Com a imagem da placa, vamos imprimir ela em um papel brilhante de fotografia (o fosco não funciona) ou em uma folha de revista utilizando uma impressora à laser (ou seja, toner; a impressora jato de tinta não funciona). É importante não deixar a impressora mudar o tamanho da imagem. Para isso, utilize sempre a função tamanho real, pois as medidas exportadas pelo *Eagle* são exatamente as dos

componentes. Abaixo mostramos como utilizar a folha de revista, já que a maior parte das impressoras não consegue imprimir na folha sozinha, por ser muito fina e pequena.

Dica: imprima no papel que você vai usar de suporte o desenho para conseguir alinhar o papel de revista

Com o desenho impresso, prendemos ele na placa, que deve ser muito bem lavada (sem manchas de gordura, por exemplo), com o toner pra baixo e utilizamos um ferro de passar roupa para transferir o toner para a placa de cobre virgem. Com o calor a tinta se solta da folha de revista ou do papel fotográfico. Não deixe o seu ferro aquecer demais, nem ficar muito tempo em cima do mesmo ponto da placa, movimente-o, como se estivesse passando a placa, senão você pode soltar o cobre da placa. Durante a transferência, não mexa muito no papel para não desalinhar o desenho.

Depois de passar a placa, coloque-a em um balde com água fria e passe a mão delicadamente por cima do papel para que ele se dissolva e solte da placa. Apenas as linhas pretas dos caminhos continuaram grudadas na placa. Lave bem a placa para tirar restos de gordura e confira se existe alguma falha no caminho, caso localize alguma, utilize uma caneta de projetor para preenchê-la.

Conferidas todas as trilhas, dissolva o percloreto de ferro (que pode ser encontrado em lojas de eletrônica em geral) em água de acordo com as instruções do produto e mergulhe sua placa na solução. O percloreto irá corroer o cobre na superfície não marcada pelo toner ou pela caneta, sobrando apenas as trilhas. Quando restar apenas os caminhos, retire a placa da solução e lave-a bem. Utilize um furador de placas para fazer os furos dos componentes e então solde-os.

Neste capítulo, explicamos de forma sucinta o processo de confecção das placas de circuito impresso. Caso deseje mais informações, detalhes e tutoriais sobre o design de esquemáticos e do desenho das trilhas, sugerimos que utilize o site *http://instructables.com*. No Instructables você encontrará diversos tutoriais relativos a utilização de diversos softwares, inclusive do Eagle, além de técnicas e macetes para o processe de impressão das placas.

7.3.2 Soldagem e Dessoldagem

Soldar é uma das mais importantes habilidades na eletrônica, sendo utilizada em praticamente qualquer projeto do ramo. Neste capítulo falaremos sobre a soldagem com várias dicas. Vale ressaltar que, como qualquer outra habilidade, somente a prática ajudará no seu desenvolvimento. A primeira coisa que vamos apresentar é o ferro de solda e a solda ou estanho:

Essas são a ferramenta e a matéria prima para a soldagem. Vamos começar com dicas de manutenção do seu ferro de solda.

- Mantenha a ponta do seu ferro estanhada e livre de sujeiras, para isso, utilize uma esponja para ferro de solda molhada.
- Evite utilizar seu ferro para derreter qualquer coisa que não seja estanho

Estanho de qualidade é algo muito importante, uma solda ruim pode ter uma resistência muito alta ou uma baixa aderência. A escolha do ferro depende da sua utilização. Para projetos de robótica educacional, principalmente soldagem em placas e união de fios, o correto é que o ferro não tenha mais que 40W de potência. Ferramentas mais potentes podem aquecer demais,

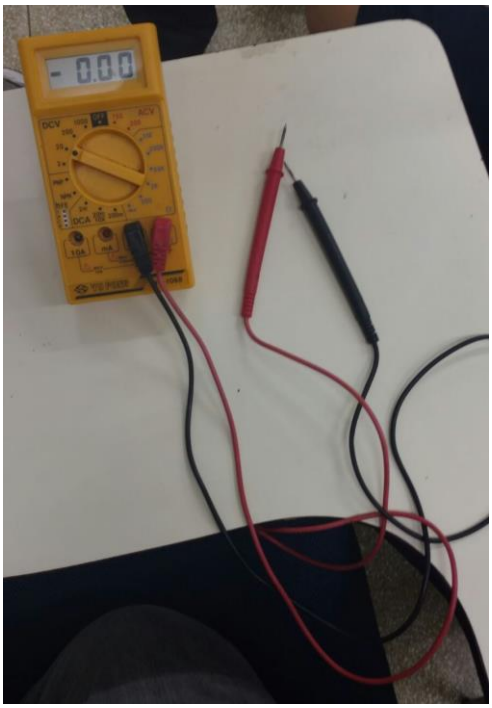
queimando componentes e danificando as placas de cobre, além de dificultar a aderência do estanho à placa.

Depois de adquirir o necessário, daremos algumas dicas para a soldagem em si:

- Garanta que as duas superfícies a serem soldadas estejam limpas, utilize uma lixa fina ou palha de aço para limpeza.
- No caso de dois fios ou terminais, faça uma ligação mecânica antes de soldar (enrolar os fios, por exemplo).
- Não derreta o estanho na ponta do ferro. Utilize a ponta para aquecer a trilha e a perna do componente ou os terminais e encoste o estanho no local aquecido (um ou dois segundos devem ser suficientes, mas retire primeiro o estanho e depois o ferro, em um movimento ascendente).
- O aspecto de uma boa solda é liso e brilhante.
- Para a soldagem de alguns componentes sensíveis ao calor como semicondutores (diodos, LED's, etc.) e capacitores eletrolíticos é interessante o uso de um alicate de bico para desviar o calor.
- Nunca ultrapasse 5 segundos com o ferro em contato com um componente. Caso não consiga neste tempo, pare e aguarde um pouco para o componente esfriar e tente novamente.
- Não utilize muita solda, somente o necessário, como diria Mogli, o menino lobo.

Para dessoldar, é importante a utilização de um sugador de solda, como o mostrado abaixo. Sua utilização é simples, abaixa-se a manivela do sugador e com o ferro de solda, derrete-se a solda a ser retirada. Com o estanho derretido, aproxima-se a ponta do sugador e aperta-se o botão. A alavanca subirá e sugará a solda.

7.3.3 Multímetro e medidas



O multímetro é um aparelho que reúne um amperímetro, um voltímetro e outros medidores elétricos em um só. É uma ferramenta extremamente importante, vamos descrever aqui os principais modos de utilização:

Voltímetro - Para medir a diferença de potencial entre dois pontos, deve-se colocar as pontas de prova paralelas ao circuito. Da mesma forma se mede a resistência do circuito.

Amperímetro - Para se medir a corrente, coloca-se as pontas de prova em série com o circuito a ser analisado. A medição em série pode queimar o aparelho.

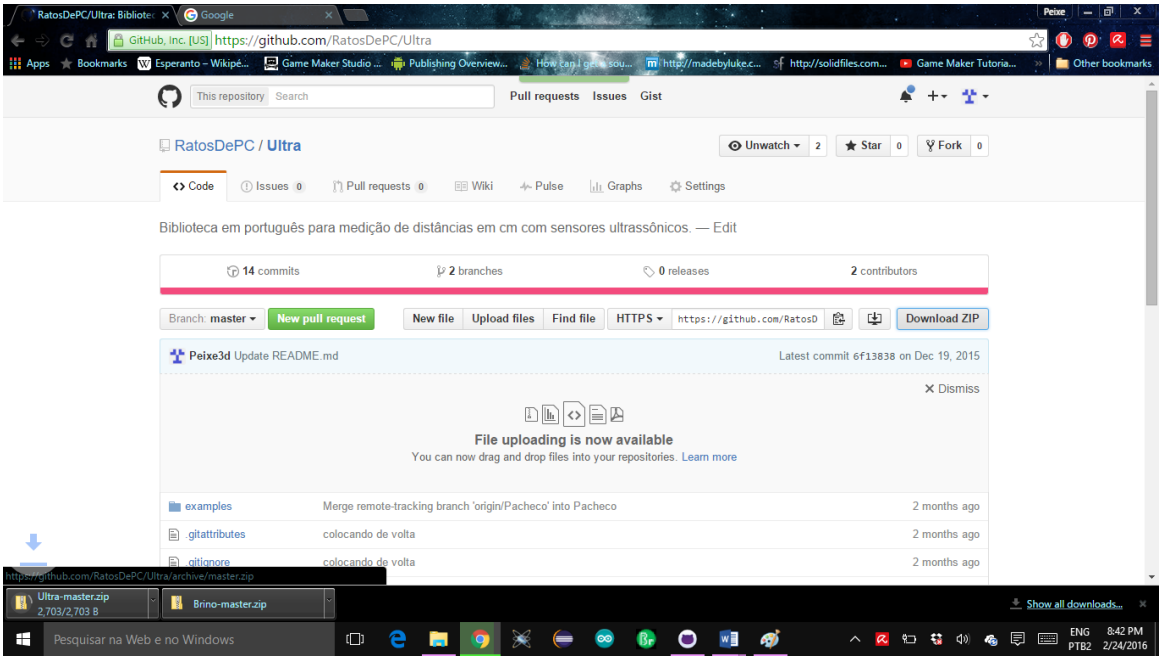
7.3.4 Como instalar bibliotecas externas

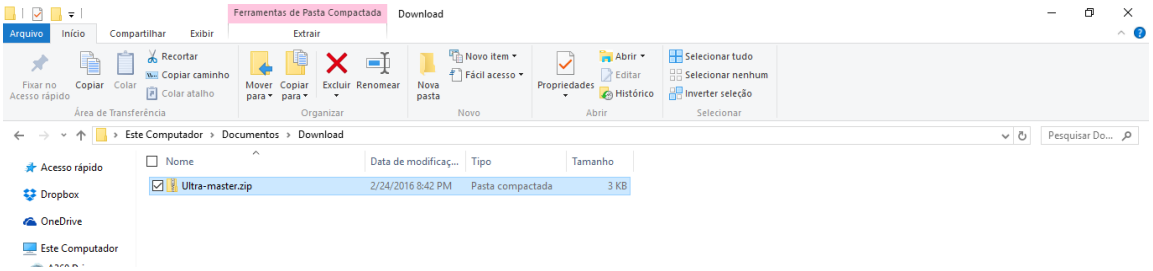
As vezes o desenvolvimento exige o uso de trechos complexos e repetitivos de código. Para evitar que “reinventemos a roda”, ou repitamos demais algumas funções pelo programa, existem as bibliotecas. Elas nada mais são do que coleções de funções úteis e comumente utilizadas

por diversos programadores. Existem diversas disponíveis na internet e você pode criar suas próprias para utilizar em diversos projetos.

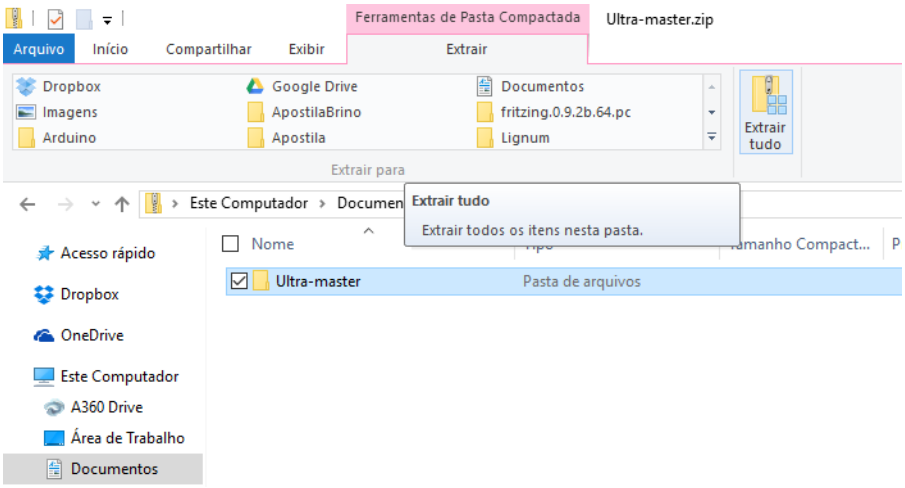
Neste capítulo mostraremos como instalar uma biblioteca baixada da internet. Para exemplificar utilizaremos a biblioteca Ultra desenvolvida pelos autores do livro e disponível no GitHub. Você pode encontrá-la em http://github.com/RatosDePC/Ultra e para baixar, basta clicar em "download zip". A instalação é bem simples, seguiremos três passos:

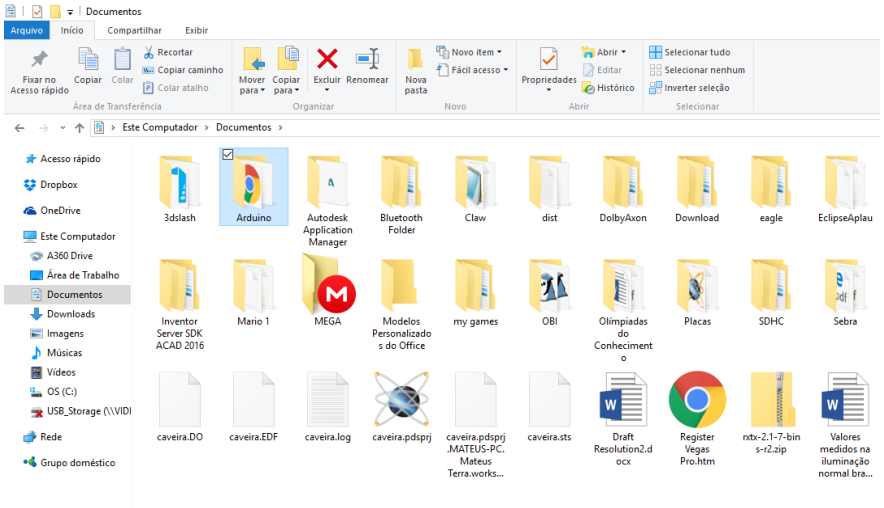
Passo 1: Baixe a biblioteca e salve em algum lugar onde você possa encontrar.





Passo 2: Extraia a biblioteca em uma pasta e copie a pasta com o nome da biblioteca (não altere o nome da pasta) para a pasta libraries do arduino, localizada em "Documentos/Arduino"


Ferramentas de Pasta Compactada
Ultra-master.zip
Arquivo
Início
Compartilhar
Exibir
Extrair
Dropbox
Imagens
Arduino
Google Drive
ApostilaBrino
Apostila
Documentos
fritzing.0.9.2b.64.pc
Lignum
Extrair tudo
Extrair para
Este Computador
Documen
Extrair tudo
Extrair todos os itens nesta pasta.
Acesso rápido
Dropbox
OneDrive
Este Computador
A360 Drive
Área de Trabalho
Documentos
Nome
Ultra-master
Pasta de arquivos


Documentos
Arquivo
Início
Compartilhar
Exibir
Este Computador
Documentos
Acesso rápido
Dropbox
OneDrive
Este Computador
A360 Drive
Área de Trabalho
Documentos
Downloads
Imagens
Músicas
Vídeos
OS (C:)
USB_Storage (\\VIDI
Rede
Grupo doméstico
3dslash
Arduino
Autodesk Application Manager
Bluetooth Folder
Claw
dist
DolbyAxon
Download
eagle
EclipseAplau
Inventor Server SDK ACAD 2016
Mario 1
MEGA
Modelos Personalizados do Office
my games
OBI
Olimpiadas do Conhecimento
Placas
SDHC
Sebra
caveira.DO
caveira.EDF
caveira.log
caveira.pdsprj
caveira.pdsprj.MATEUS-PC.Mateus Terra.works...
caveira.sts
Draft Resolution2.docx
Register Vegas Pro.htm
rxtx-2.1-7-bins-r2.zip
Valores medidos na iluminação normal bra...

Referências Bibliográficas

MCROBERTS, M. **Arduino Básico**, Quarta reimpressão. São Paulo: Novatec, 2013. 453 p.

Evans M.; Noble, J.; Hochenbaum, J. **Arduino em ação**, primeira reimpressão. São Paulo: Novatec, 2014. 424 p.

http://arduino.cc

http://fritzing.org

http://instructables.com

http://cadsoftusa.com