UNIVERSIDADE FEDERAL DE UBERLÃNDIA-UFU
FACULDADE DE ENGENHARIA MECÂNICA-FEMEC
ENGENHARIA MECATRÔNICA

JEAN ROBERT DA CUNHA MARQUEZ
YURI LIMA ALMEIDA

APOSTILA CURSO DE ARDUINO

UBERLÂNDIA
2017

JEAN ROBERT DA CUNHA MARQUEZ
YURI LIMA ALMEIDA

ATIVIDADES EXTRACURRICULARES PRÁTICAS UTILIZANDO INFRAESTRUTURA DOS LABORATÓRIOS DE ENSINO PARA INGRESSANTES

Relátorio apresentado ao curso de Engenharia Mecatrônica,ESCOLA,CIDADE,para fins...

Orientadora: Profa. Dra Vera Lúcia Donizeti de Sousa Franco

UBERLÂNDIA
2017

SUMÁRIO

INTRODUÇÃO

"Um Arduino é uma plataforma de prototipagem eletrônica de placa única, projetada com um microcontrolador Atmel AVR com suporte de entrada/saída embutido, uma linguagem de programação padrão, a qual tem origem em Wiring, e ***é essencialmente C/C++****. O objetivo do projeto é* ***criar ferramentas que são acessíveis, com baixo custo, flexíveis e fáceis de se usar por artistas e amadores****. Principalmente para aqueles que não teriam alcance aos controladores mais sofisticados e de ferramentas mais complicadas".*

Figura 1: Vista superior de um Arduino Uno.

Na prática, temos que um Arduino é um pequeno computador que pode ser programado para processar entradas e saídas entre o dispositivo e os componentes externos conectados a ele. O Arduino é um exemplo de uma plataforma de computação física ou embarcada, ou seja, um sistema que pode interagir com seu ambiente por meio de *hardware* e *software.*

Para programar o Arduino, é necessário seu IDE (*Integrated Development Environment*), um *software* livre no qual o código escrito é processado e compreendido pelo Arduino. Os códigos escritos nesse *software* são conhecidos como *sketches* (rascunho, esboço).

Tanto o *software* quanto o *hardware* do Arduino são ambos de fonte aberta, o que significa que os códigos, projetos e esquemas podem ser utilizados livremente por qualquer pessoa e com qualquer propósito.

Existem diversos tipos de Arduino, sendo o mais versátil, acessível e mais utilizado o Arduino Uno, mostrado na figura acima. Por ser de fonte aberta, existem diversos clones do Arduino, todos compatíveis com o IDE disponibilizado. Esses clones são totalmente permitidos, com a única restrição de que não tenham o nome "Arduino", dado ao original.

A placa do Arduino é composta de um microprocessador Atmel AVR, um cristal ou oscilador (relógio simples que envia pulsos de tempo em uma frequência especificada, para permitir sua operação na velocidade correta) e um regulador linear de 05 volts. Essa placa expõe os pinos de entrada/saída do microcontrolador, de modo a conectá-los facilmente a outros circuitos ou sensores.

1. INICIANDO NO ARDUINO:

1.1.VANTAGENS:

Facilidade de utilização;

Open-source (código aberto);

Grande comunidade de pessoas que utilizam Arduinos e compartilham seus códigos e diagramas de circuito para que outros os copiem e modifiquem;

Preço acessível.

1.2. INSTALAÇÃO:

Para começar a desenvolver seus próprios projetos em Arduino, primeiramente é necessário fazer o *download* do IDE em *h tt ps://w ww.arduino.cc/en/Main/ Soft w are*

. O *download* é gratuito e leve.

Não obrigatório, mas muito conhecido, também há o programa em ambiente gráfico ***Fritzing***, que mostra de modo virtual como seria a montagem física em *protoboard*, assim como o circuito elétrico do projeto em Arduino desenvolvido. O *link* para *download* é *htt p://fr it zing.org/dow nload/* .

1.3. SELEÇÃO DE PLACA E PORTA:

Antes de começar a programar em Arduino, é necessário selecionar corretamente a placa e a porta. Assim que o IDE for aberto, selecionamos **Menu → Tools → Board**, e selecionamos a placa a ser utilizada. No caso, clicamos em **Arduino Uno**, confirmando a placa que utilizaremos.

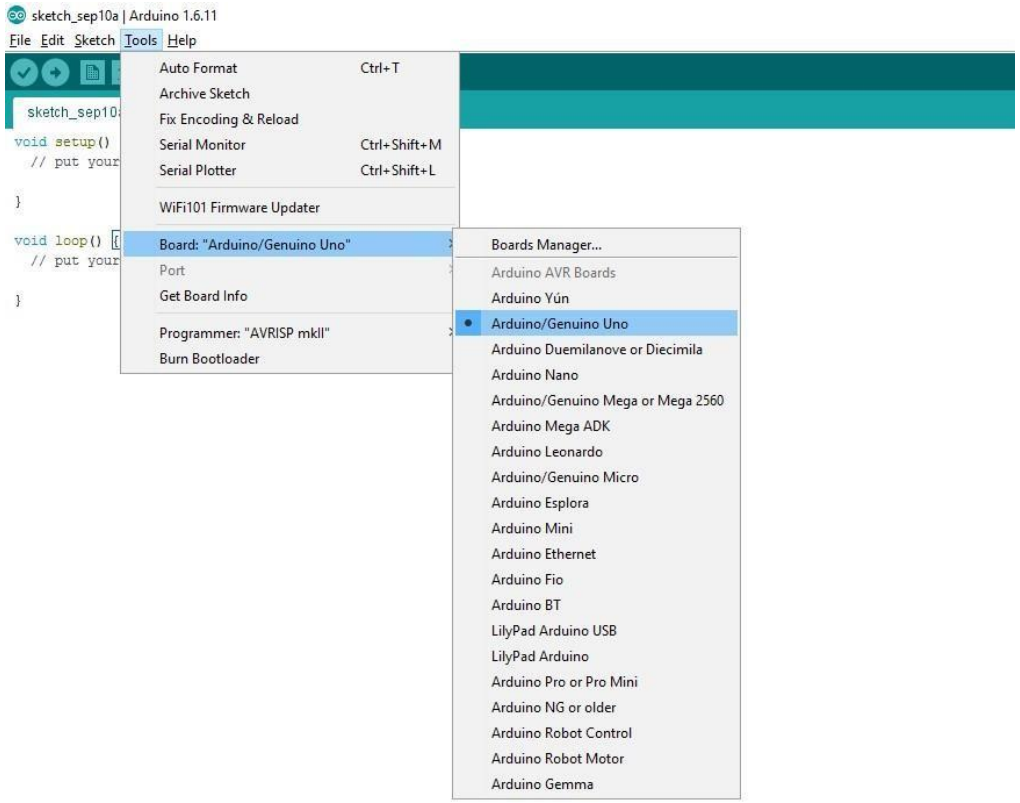

Figura 2: IDE do Arduino, na seleção da placa no menu Board

Agora, para selecionar a porta, clicamos em **Menu → Tools → Serial Port**, e selecionamos a porta adequada ao Arduino a ser utilizado.

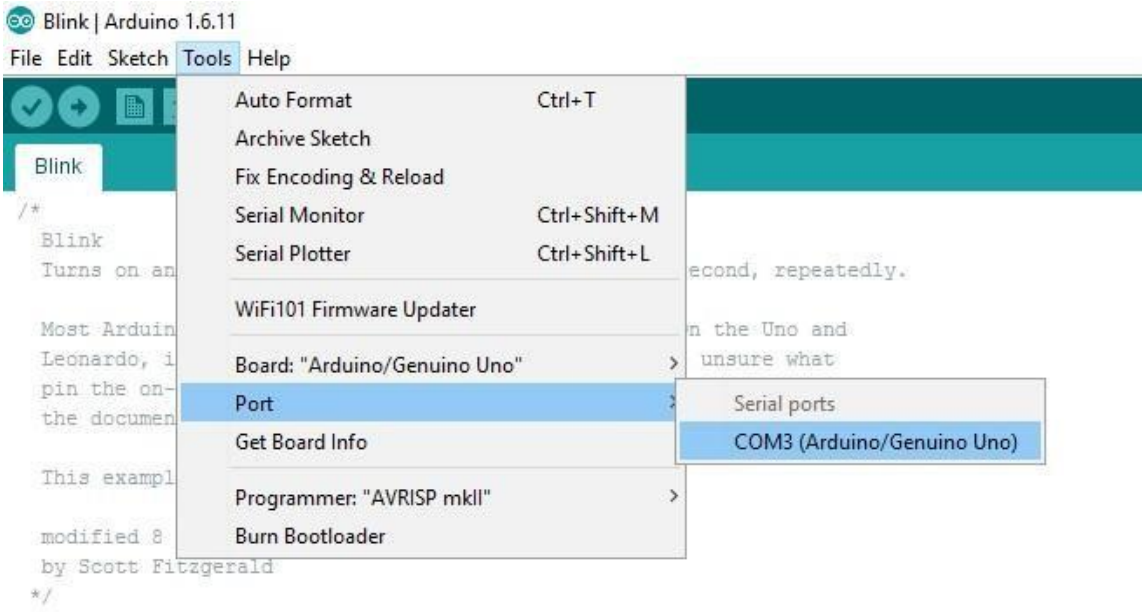

Figura 3: IDE do Arduino, na seleção da porta na lista Serial Port

1.4. ABA “EXAMPLES”:

Uma das maiores utilidades do Arduino são seus exemplos prontos. Selecionando o menu **File → Examples**, encontramos exemplos prontos, que abrangem desde o básico, até alguns recursos mais avançados como comunicação, EEPROM, relação mestre-escravo, entre outros.

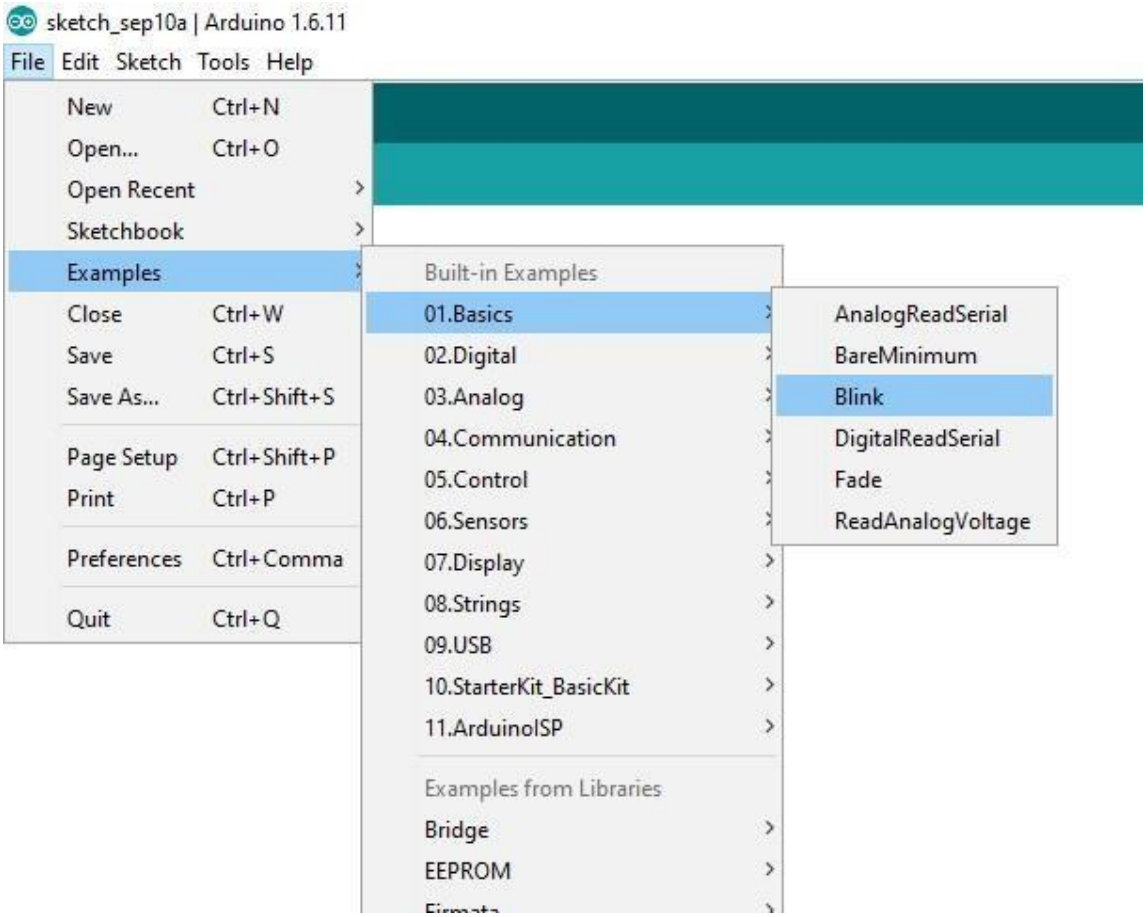

Figura 4: Menu Examples

Caso o menu *Examples* não tenha resolvido seus problemas, temos ainda a aba ***Help*** e os fóruns do Arduino na internet, que contam com uma imensa comunidade muito disposta a ajudar.

1.5. AMBIENTE DE TRABALHO:

Logo ao abrir o IDE pela primeira vez, notamos que há duas funções principais no *sketch*: *setup* e *loop*. Como explicado ao abrir, a função *setup* será executada uma vez somente, e nela deve conter toda configuração do programa; e a função *loop* será executada repetidamente, onde teremos o código principal que controla o que iremos desenvolver com o Arduino.

Vale lembrar que o código no Arduino é executado na ordem em que é disposto nas linhas do programa, e não simultaneamente.

O IDE é dividido em três partes: a *Toolbar* no topo, o código (também chamado de *Sketch Window*) no centro, e a janela de mensagens na base. Ao longo do topo, temos a barra de menus, com os itens **File, Edit, Sketch, Tools** e **Help**.

A *Toolbar* consiste de seis botões, que fornecem rápido acesso às funções utilizadas mais frequentemente dentro desses menus. Elas são, respectivamente:

Figura 5: Toolbar

Tabela 1: Funções dos botões da Toolbar.

Verify	Verifica se há erros no código
Upload	Faz o *upload* do *sketch* atual para o Arduino
New	Cria um *sketch* em branco
Open	Mostra uma lista de *sketches*, em seu *Sketchbook*, para abrir
Save	Salva o *sketch* atual em seu *Sketchbook*
Serial Monitor	Exibe os dados seriais enviados do Arduino

Note que os cinco primeiros botões são localizados logo abaixo da barra de menus, enquanto o último (Serial Monitor) fica localizado no canto superior direito.

Os nomes dos botões já são, em si, autoexplicativos. Entretanto, não custa nada criar desde o começo bons hábitos ao se programar em Arduino. Alguns principais são sempre compilar o código (função *Verify/Compile*) e salvá-lo antes de enviá-lo (função Upload) para a placa, evitando assim que um eventual erro que trave seu sistema ou o IDE gere perda no que foi desenvolvido.

Quando um erro for identificado no processo de verificação, uma mensagem de erro em texto vermelho aparecerá no canto inferior do IDE, com seu respectivo número de localização. Assim, é possível localizá-lo rapidamente e corrigi-lo da forma mais adequada possível. Um método fácil para se encontrar o erro em um código mais extenso é utilizando a seta para baixo (↓) até encontrá-lo, ou utilizando a aba **Edit**.

Caso o seu código não esteja sendo executado após verifica-lo, salvá-lo e enviá-lo à placa, verifique no menu *Tools* se a placa e a porta estão selecionadas corretamente. É um erro muito comum ao se iniciar em Arduino, então não se preocupe.

É bom perceber que após realizar um *upload*, as luzes RX e TX devem começar a piscar, mostrando que dados estão sendo transmitidos de seu computador para a placa. Assim que o *upload* do *sketch* tiver sido concluído, a mensagem *"Done uploading"* será exibida na barra de status do IDE e as luzes RX e TX pararão de piscar.

Na aba **File → Preferences**, é possível ajustar suas preferências ao se utilizar o Arduino, como, por exemplo, a localização padrão do *sketchbook*.

1.6. MONITOR SERIAL:

Entre as funções apresentadas no tópico anterior, será explicada resumidamente a função do monitor serial. O monitor serial é uma ferramenta muito útil, especialmente para depuração de código, ou seja, corrigir eventuais erros. Ele exibe os dados seriais enviados do Arduino (USB ou placa serial), assim como também permite enviar dados de volta ao Arduino pelo próprio monitor serial. Para abri-lo, basta clicar no botão ***Serial Monitor***.

Figura 6: Janela serial do Arduino

Na caixa de texto em branco na parte superior do monitor serial, é possível digitar o texto a ser enviado para o Arduino. É bom lembrar que tanto o monitor serial quanto o Arduino não podem receber nenhum dado serial, a menos que o código do *sketch* esteja preparado para entender essa informação e codificá-la.

A área central da janela é o local em que os dados seriais serão exibidos, onde é possível verificar tudo o que foi enviado e recebido pelo Arduino.

No canto inferior direito, há o chamado *Baud Rate*, que é a taxa de transmissão na qual os dados seriais devem ser enviados de/para o Arduino. A taxa de transmissão é a taxa por segundo em que alterações de estado ou *bits* (dados) são enviados de/para a placa. Por exemplo, a configuração padrão é 9600 baud, ou seja, se precisássemos enviar um livro pela linha de comunicação serial (nesse caso, USB), conseguiríamos enviar 1200 letras ou símbolos de texto por segundo (9600 bits / 8 bits por caractere = 1200 bytes ou caracteres) para esta taxa.

O exemplo anterior mostra grande parte da utilidade do monitor serial. Mais à frente trabalharemos em específico com ele, criando funcionalidades bem interativas para seu uso e aprendizado.

Quando quiser interromper o monitor serial, basta fechá-lo.

1.7. BIBLIOTECAS E SHIELDS:

"Biblioteca é uma coleção de subprogramas utilizados no desenvolvimento de software. Biblioteca contém códigos e dados auxiliares, que provém serviços a programas independentes, o que permite o compartilhamento e a alteração de código e dados de forma modular"[2].

Em linhas gerais, a biblioteca permitirá incluir em seu projeto funcionalidades adicionais, sem a necessidade de inventar um código para uma utilização já feita. Como exemplo, a biblioteca *Stepper* é um conjunto de funções para controlar um motor de passo, permitindo a reutilização do código diversas vezes em projetos diferentes e de modo simples.

Já os *Shields* são placas que se acoplam à placa original, agregando funcionalidades adicionais ao Arduino. Deste modo, o leque de possibilidades de funções que o Arduino desempenha é imenso, e por isso ele é tão utilizado, tanto para aprendizado quanto para alguns projetos.

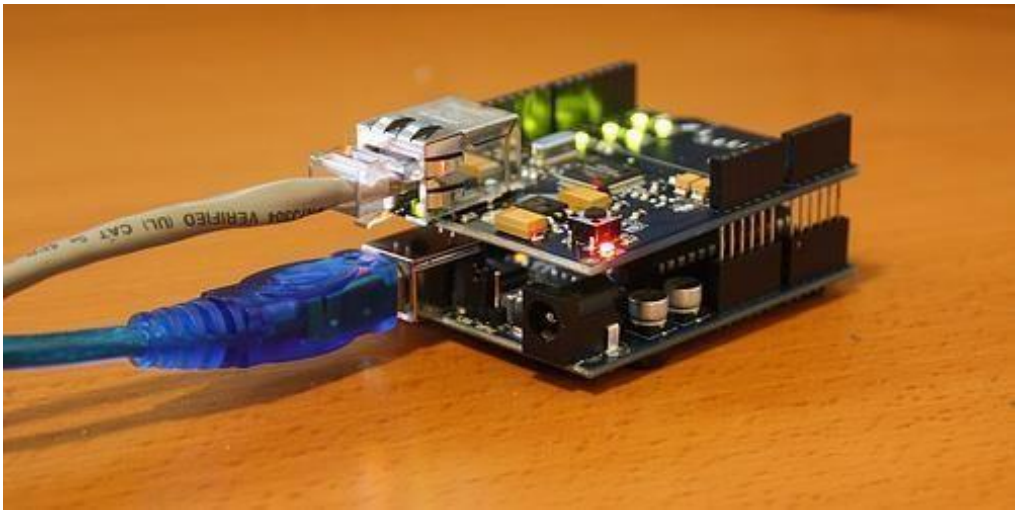

Figura 7: Exemplo de um Arduino com um shield Ethernet

2. INTEGRAÇÃO DO ARDUINO COM O COMPUTADOR:

Apesar de o Arduino ser um pequeno computador independente, ele consegue conversar com outro computador (PC) através da porta USB. Isso nos permite desenvolver um *software* que funcione em nosso PC e comunica-se com o *software* do Arduino, permitindo, assim, a integração de vários *softwares* diferentes e aumentando sua polivalência, ou seja, conseguimos resolver problemas ainda mais complexos unindo nosso conhecimento básico de Arduino com alguns *softwares* específicos para certas finalidades.

3. PROTOBOARD:

*"Uma placa de ensaio ou matriz de contato (**protoboard**, ou breadboard em inglês) é uma placa com furos e conexões condutoras para montagem de circuitos elétricos experimentais. A grande vantagem da placa de ensaio na montagem de circuitos eletrônicos é a **facilidade de inserção de componentes**, uma vez que não necessita soldagem. As placas variam de 800 furos até 6000 furos, tendo conexões horizontais e verticais"[3].*

Uma vez compreendido o que é uma ***protoboard***, passaremos a entender seu funcionamento prático. Na parte central, onde se concentram a maior parte dos furos, temos várias tiras metálicas, como se observa na figura a seguir:

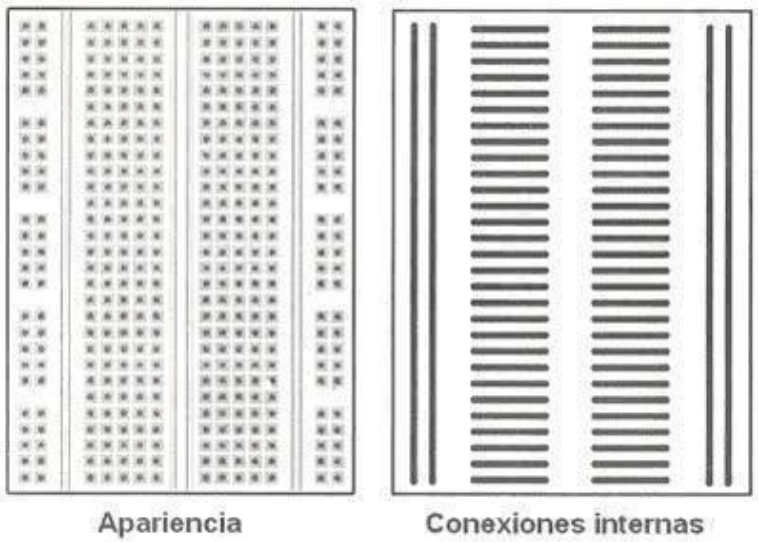

Figura 8: Aparência de uma protoboard e suas conexões internas

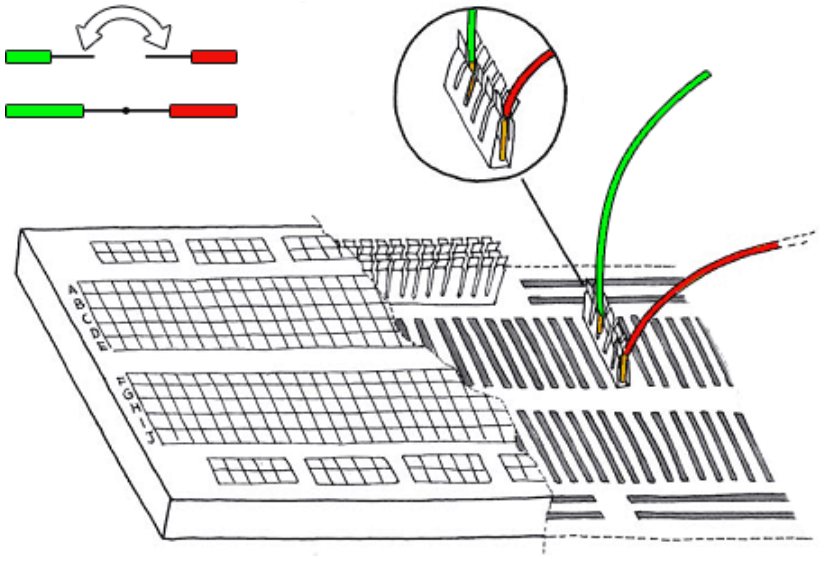

Figura 9: Esquema de como funciona a conexão aos furos da protoboard

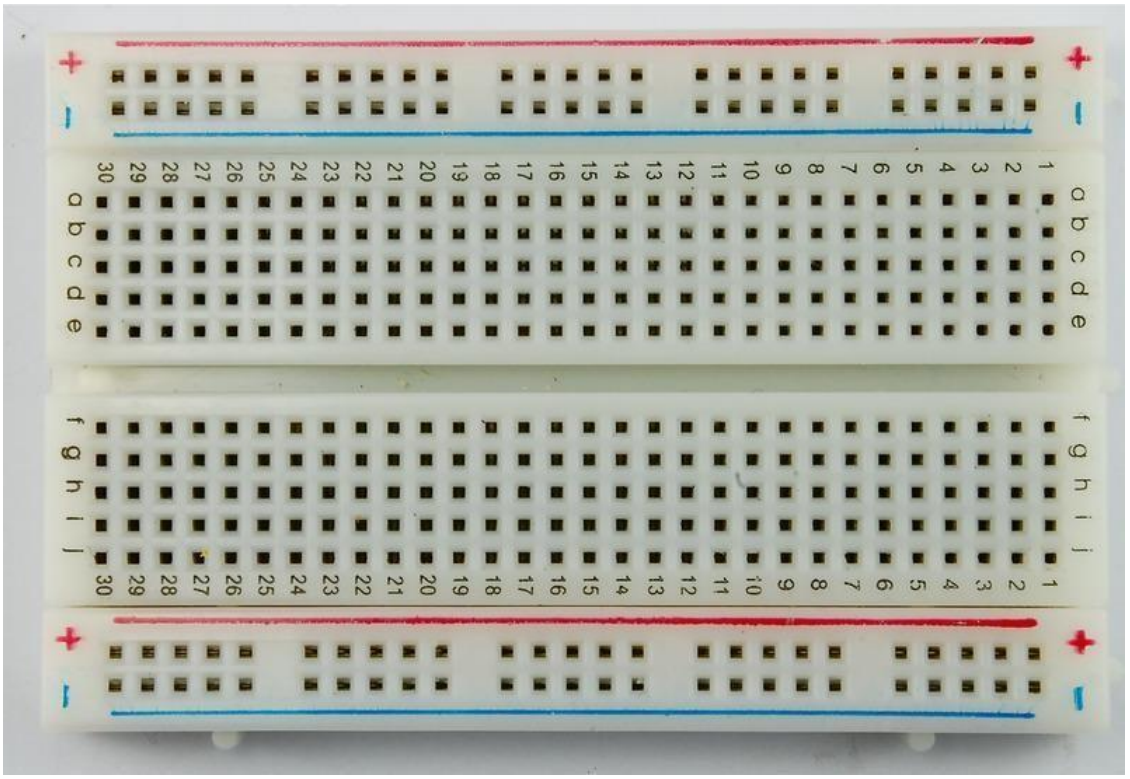

Figura 10: Um exemplo de protoboard

O resultado disso é uma superfície equipotencial, ou seja, pontos que estejam sobre a mesma tira metálica possuem o mesmo potencial elétrico. Na prática, isso nos permite e facilita a montagem do circuito, uma vez que temos 05 furos (no caso da *protoboard* em questão) a que podemos colocar um conector para obter o potencial desejado.

Geralmente, conecta-se a alimentação da bateria (05 volts, ou a tensão qualquer a ser utilizada) e o Terra (*GND*, ou *ground*, em inglês) nas tiras metálicas dos cantos, de forma a facilitar a visualização e montagem do circuito.

Inicialmente pode parecer confuso, mas assim que os primeiros projetos, tudo fará muito sentido e entendido de forma simples. Assim, vamos começar a parte de programação em Arduino e, em seguida, montar os circuitos para se examinar o funcionamento do código.

4. PROGRAMAÇÃO BÁSICA:

Antes de começar a programar seu primeiro projeto, vamos ver algumas funções do Arduino, assim como seus operadores lógicos mais simples, de forma a abrangermos grande parte da lógica de programação necessária para tal fim.

5.1. CONSTANTES BOOLEANAS:

Na linguagem do Arduino, são três os grupos de constantes booleanas. Os componentes de cada grupo apresentados a seguir podem ser representados, respectivamente, pelos números binários 1 e 0.

TRUE / FALSE : definem os estados lógicos. Verdadeiro é qualquer valor diferente de zero, enquanto Falso é sempre o valor zero;
HIGH / LOW : definem as tensões nos pinos digitais do Arduino. Alto é uma tensão de 05 volts, e Baixo (ou Terra) é uma tensão de 0 volts;
INPUT / OUTPUT : são constantes programadas pela função ***pinMode*** para os pinos do Arduino. Elas podem ser entradas (sensores) ou saídas (de controle).

5.2. VARIÁVEIS:

As variáveis são posições na memória do programa Arduino, identificadas por nomes e tipos de informação a serem guardadas. Elas podem ser declaradas tanto vazias, quanto com um valor inicial, assim como é feito na programação em linguagem *C*.

BYTE : armazena 8 *bits* (0-255, 2^8 binários);

INT : armazena números inteiros de até 16 *bits*;
LONG : armazena números inteiros de até 32 *bits*;
FLOAT : armazena números fracionários de até 32 *bits;*

CHAR : armazena um tipo de dado qualquer que ocupe 1 *byte* de memória;
STRING : armazena um vetor de dados do tipo *'char'*.

Além disso, podemos também colocar a declaração *unsigned* na variável, de modo a só apresentar valores "positivos" (sem sinal).

5.3. OPERAÇÕES ARITMÉTICAS:

Tabela 2: Operadores aritméticos usuais

x ++	x = x + 1
x --	x = x -1
x += y	x = x + y
x -= y	x = x – y
x *= y	x = x*y
x /= y	x = x/y

5.4. OPERADORES DE COMPARAÇÃO:

Tabela 3: Operadores de comparação usuais

x == y	x é igual a y
x != y	x não é igual a y
x < y	x é menor que y
x > y	x é maior que y
x <= y	x é menor ou igual a y
x >= y	x é maior ou igual a y

5.5. OPERADORES LÓGICOS:

Tabela 4: Operadores lógicos usuais

&&	AND	Porta lógica “E”
||	OR	Porta lógica “OU”
!	NOT	Porta lógica “NÃO”

5. FUNÇÕES:

Aqui, serão apresentadas algumas funções tradicionais, já conhecidas da programação na linguagem *C/C++*, e, logo em seguida, algumas funções a fim de se tratar especificamente com o Arduino.

6.1. FUNÇÕES COMPARATIVAS E CONDICIONAIS:

if :

```
if (expressão){

Bloco de instruções; // executa se "expressão" for verdadeira

}
```

if ... else :

```
if (expressão){

Bloco de instruções; // executa se "expressão" for verdadeira

}

else {

Bloco de instruções 2; // executa se "expressão" for falsa

}
```

switch ... case :

```
switch (expressão){

case 1: // bloco de instruções 1 break;
case 2: // bloco de instruções 2 break;
default: // bloco de instruções 3

}
```

while : while (expressão){
bloco de instruções; // executa enquanto a “expressão” for verdadeira

}

for :

```
for (variável; expressão; incremento){
Bloco de instruções; // executa enquanto a "expressão" for verdadeira
}
```

6.2. FUNÇÕES DE TEMPO E ESPERA:

delay(ms) : pausa o programa, em milissegundos, de acordo com o parâmetro entre parênteses. Durante o funcionamento dessa função, qualquer outra função no programa é suspensa;

millis() : retorna o tempo (em milissegundos) que se passaram desde que o Arduino foi ligado. O retorno deve ser armazenado em uma variável do tipo *long*.

6.3. FUNÇÃO MATEMÁTICA DE NÚMERO ALEATÓRIO:

random(min, max) : gera números aleatórios entre os limites mínimo e máximo especificados como parâmetros. Caso o parâmetro do limite mínimo seja excluído, será atribuído como padrão 0.

6.4. FUNÇÕES PARA CONFIGURAR AS PORTAS DE ENTRADA E SAÍDA:

pinMode(pino, modo) : Essa função é **sempre escrita dentro da função *void setup()***. Ela estabelece o fluxo de informações nos pinos digitais do Arduino. O primeiro parâmetro é o número do pino a ser utilizado, e o segundo é sua configuração como entrada (modo *INPUT*) ou saída (modo *OUTPUT*);

digitalRead(pino) : O pino usado como parâmetro deve estar configurado como uma entrada. Essa função realiza a leitura do estado lógico (verdadeiro ou falso) do pino em questão e a armazena em uma variável (*HIGH* ou *LOW*);

digitalWrite(pino, valor) : O pino usado como parâmetro deve estar configurado como uma saída. Essa função envia um nível lógico para o pino digital em questão;

analogRead(pino) : Realiza a leitura do nível analógico do pino, armazenando este dado em uma variável definida pelo programador.

Vale ressaltar-se que os pinos 0 a 13 são pinos digitais (sendo os pinos 0 e 1 indicados para, respectivamente, RX e TX), e os pinos A0 a A5 são pinos analógicos.

6.5. FUNÇÕES PARA COMUNICAÇÃO SERIAL (PORTAS RX E TX):

Seria.begin(taxa) : define a taxa de transferência, em *bits* por segundo. A taxa de transferência padrão é de 9600;

Serial.println() : imprime dados na porta serial como texto ASCII e pula, em seguida, uma linha;

Serial.availabe() : verifica o número de *bytes* (caracteres) disponíveis para leitura na porta serial;

Serial.read() : realiza a leitura dos dados que chegam à porta serial;

Serial.write() : escreve os dados binários na porta serial;

Serial.flush() : espera até que a transmissão de dados acabe e, em seguida, limpa o *buffer*.

1. LED’S

O led é um diodo emissor de luz, além disso é um componente polarizado e deve ser corretamente conectado. Perceba na Figura que o polo positivo possui uma perna maior. Alguns Leds apresentam o polo negativo com um chanfro (parte ligeiramente plana).

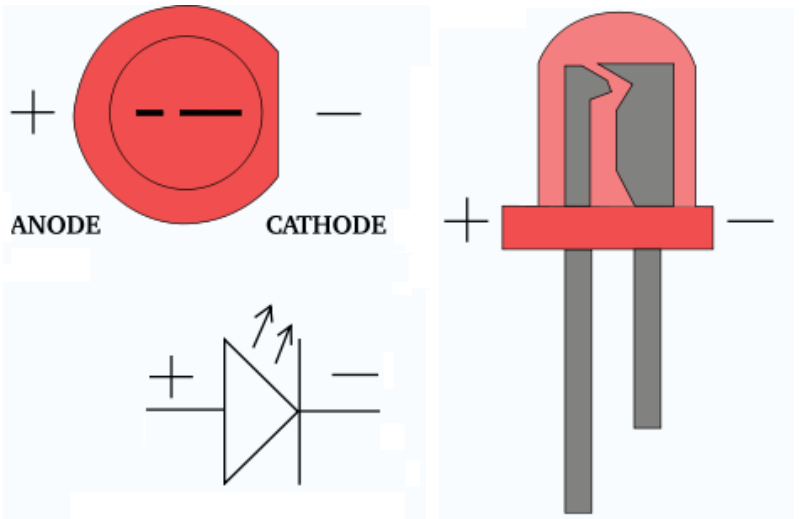

Figura 11: Esquemático de um LED.

A melhor maneira de entender o funcionamento das luzes LED’s no arduino é entendo o código deste tópico. Como projeto faremos um semáforo simples utilizando algumas funções básicas do arduino discutidos do tópico anterior. Como a tensão dos leds é de 2V e o pino digital do arduino libera 5V é recomendável a utilização de um resistor para reduzir esta tensão.

Código do projeto:

```
const int verde = 7;                    // led verde no pino 7
const int amarelo = 4;                  // led amarelo no pino 4
const int vermelho = 2;                 // led vermelho no pino 2

void setup() {

pinMode(verde, OUTPUT);         // define verde como saída
pinMode(amarelo, OUTPUT);               // define amarelo como saída
pinMode(vermelho, OUTPUT);              // define vermelho como saída

}
void loop() {

digitalWrite(verde, HIGH);      // acende o verde e apaga os demais
digitalWrite(amarelo, LOW);
digitalWrite(vermelho, LOW);

delay(25000);                   // tempo em milissegundos a esperar = 25s
```

```
digitalWrite(verde, LOW);              // acende o amarelo e apaga os demais
digitalWrite(amarelo, HIGH);
digitalWrite(vermelho, LOW);

delay(3000); // espera 3s

digitalWrite(verde, LOW);              // acende o vermelho e apaga os demais
digitalWrite(amarelo, LOW);
digitalWrite(vermelho, HIGH);

delay(12000); // espera 12s
}
```

Montagem:

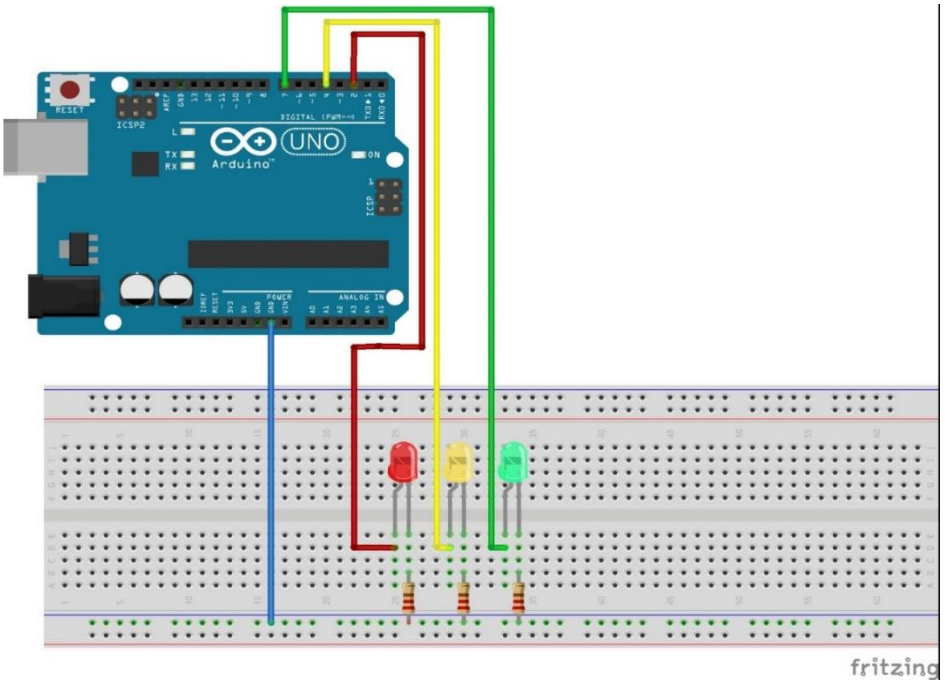

Figura 12: Montagem do projeto.

Desafio 1: Crie uma mensagem de S.O.S luminosa em código morse, lembrando que a letra S é representada por três pontos e a letra O é representada por três traços.

Desafio 2: Crie um semáforo que tenha tanto o semáforo para veículos quanto para pedestres.

2 – POTENCIÔMETRO:

FUNCIONAMENTO DO POTENCIÔMETRO E SERVO MOTORES;

Projeto: Dimmer, Controlando um servo motor;

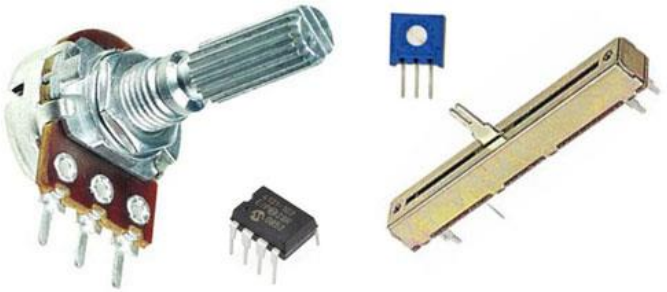

Figura 13: Um exemplos de potenciômetros

Potenciômeto é um componente eletrônico que cria uma limitação para o fluxo de corrente elétrica que passa por ele, e essa limitação pode ser ajustada manualmente, podendo ser aumentada ou diminuida. Os potenciômetros e o resistores tem essa finalidade de limitar o fluxo de corrente elétrica em um circuito, a diferença é que o potenciômetro pode ter sua resistência ajustada e o resistor comum não pode pois ele possui um valor de resistência fixo.

Figura 14: Tipo de ligação potenciômentro.

Ponteciômetro 1: está com os terminais 1 e 2 ligados, neste caso ele varia sua resistência entre 0 ohm e 10 k ohms, nessa ligação quando você gira o eixo para a esquerda ele diminui a sua resistência e quando você gira para a direita aumenta a sua resistência.

Ponteciômetro 2: está com os terminais 2 e 3 ligados, neste caso ele varia sua resistência entre 0 ohm e 10 k ohms, nessa ligação quando você gira o eixo para a esquerda ele aumenta a sua resistência e quando você gira para a direita diminui a sua resistência.

Ponteciômetro 3: a resistência é fixa, no caso 10 k ohms. Mesmo se você girar o eixo para qualquer lado a resistência não varia.

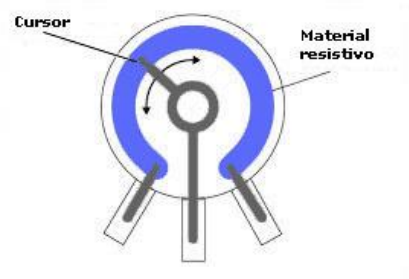

Figura 15: Funcionamento interno de um potenciômetro.

DIMMER:

Código do projeto:

```
int pinoled = 10;  //Pino ligado ao anodo do led
int pinopot = 5;   //Pino ligado ao pino central do potenciometro
int valorpot = 0;  //Armazena valor lido do potenciometro, entre 0 e 1023

float luminosidade = 0; //Valor de luminosidade do led

void setup()
{
   Serial.begin(9600);        //Inicializa a serial
   pinMode(pinoled, OUTPUT);  //Define o pino do led como saída
   pinMode(pinopot, INPUT);   //Define o pino do potenciometro como entrada
}

void loop()
{
  // Le o valor - analogico - do pino do potenciometro
  valorpot = analogRead(pinopot);

  //Converte e atribui para a variavel "luminosidade" o
  // valor lido do potenciometro
  luminosidade = map(valorpot, 0, 1023, 0, 255);

  Serial.print("Valor lido do potenciometro : ");

  //Mostra o valor lido do potenciometro no monitor serial
  Serial.print(valorpot);
  Serial.print(" = Luminosidade : ");

  //Mostra o valor da luminosidade no monitor serial
  Serial.println(luminosidade);

  //Envia sinal analogico para a saída do led, com luminosidade variavel
  analogWrite(pinoled, luminosidade);
}
```

SERVO MOTOR:

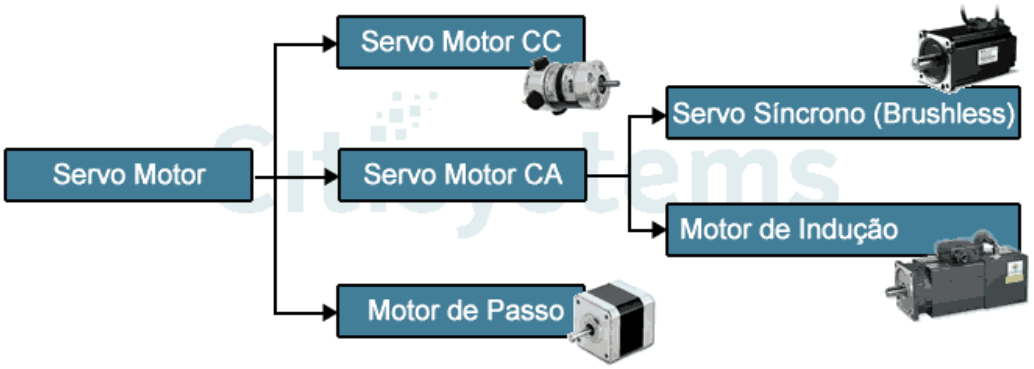

Figura 16: Tipos de servo motor.

Característica	**Moto Motor**	**Servomotor CC**	**Servomotor CA Síncrono**	**Servomotor CA Indução**
Capacidade (watts)	– Menor de 100W	– Menor do que 500W	– De 100 a 3,5 KW	– Acima de 3,5 KW
Vantagens	– Compacto – Custo reduzido	– Pequena dimensão externa – Alto torque – Boa eficiência e controle -Custo acessível	– Alta Velocidade – Alto torque – Boa eficiência operacional – Baixa manutenção	– Alta Velocidade – Altos picos de torque – Boa eficiência operacional – Baixa manutenção – Durabilidade
Desvantagens	– Ruído magnético – Baixa velocidade	– Limite na retificação – Baixa confiabilidade – Maior manutenção	– Custo alto	– Baixa eficiência em capacidades menores – Controle complexo – Custo elevado

Potenciômetro Controlando Servo Motor

Código do projeto:

```
#include "Servo.h"

// Criar um Objeto Servo
Servo servo1;

void setup()
```

```
{
  // Anexa o Servo ao Pin5
  servo1.attach(5);
}

void loop()
{
  // Lê o valor do Potenciometro
  int angle = analogRead(0);
  // Mapeia o valor de 0 a 180 graus
  angle=map(angle, 0, 1023, 0, 180);
  // Repassa o angulo ao ServoWrite
  servo1.write(angle);
  // Delay de 15ms para o Servo alcançar a posição
  delay(15);
}
```

Montagem:

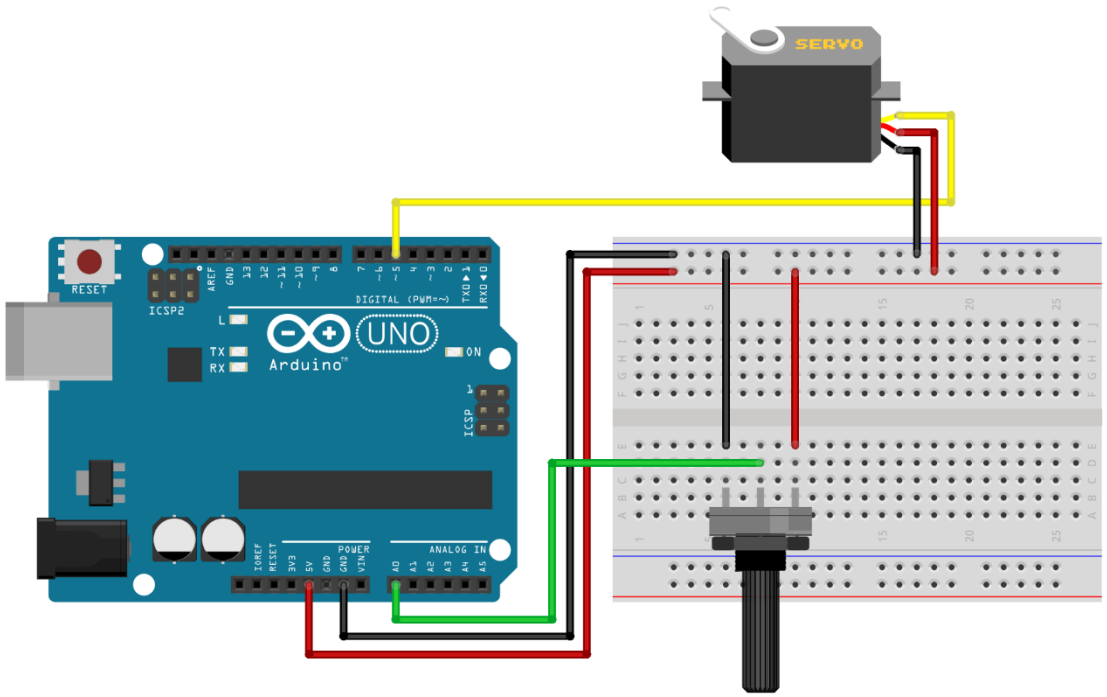

Figura 17: Montagem do projeto.

Desafio: Controle da frequência de pulso de um led.

3. SENSORES BÁSICOS

Trabalhar com sensores no arduino não é um desafio quando se tem em mãos a biblioteca do sensor específico assim como o seu Datasheet. No caso do nosso curso não serão utilizados sensores de muita complexidade. Quando se está lidando com sensores a utilização do monitor serial é no mínimo crucial. Neste Tópico serão utilizados como objetos de estudo o sensor LM35 para medir temperatura e um LDR para medir luminosidade.

LM35 (SENSOR DE TEMPERATURA)

Este sensor funciona como um transistor variando sua tensão de saída através do pino *vout* em função da temperatura em sua superfície tendo nesta saída um sinal de 10mV para cada Grau Celcius de temperatura. Além disso, sua faixa de temperatura varia entre -55°C a 150°C e a tensão a qual é operado varia de 4V à 20V.

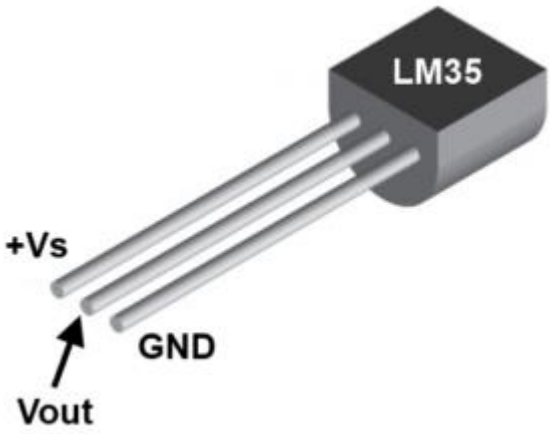

Figura 18: Sensor LM35 .

O Projeto que usaremos de exemplo para este sensor é de um termômetro simples que mostra no monitor serial a temperatura lida no ambiente. Para esse projeto apenas será necessário o arduino e o sensor LM35. O calculo presente neste código se refere tensão liberada pelo transistor e transformada em temperatura.

Código da aula:

```
const int LM35 = A0;                // Define o pino que lera a saída do LM35
float temp;                         // Variável que armazenará a temperatura medida
```

```
 void setup() {
Serial.begin(9600);            // inicializa a comunicação serial
}
//Função que será executada continuamente
void loop() {
temp = ( float (analogRead(LM35)) *5/(1023))/0.01;
Serial.print("Temperatura: ");
Serial.println(temp);
delay(2000);
}
```

Montagem:

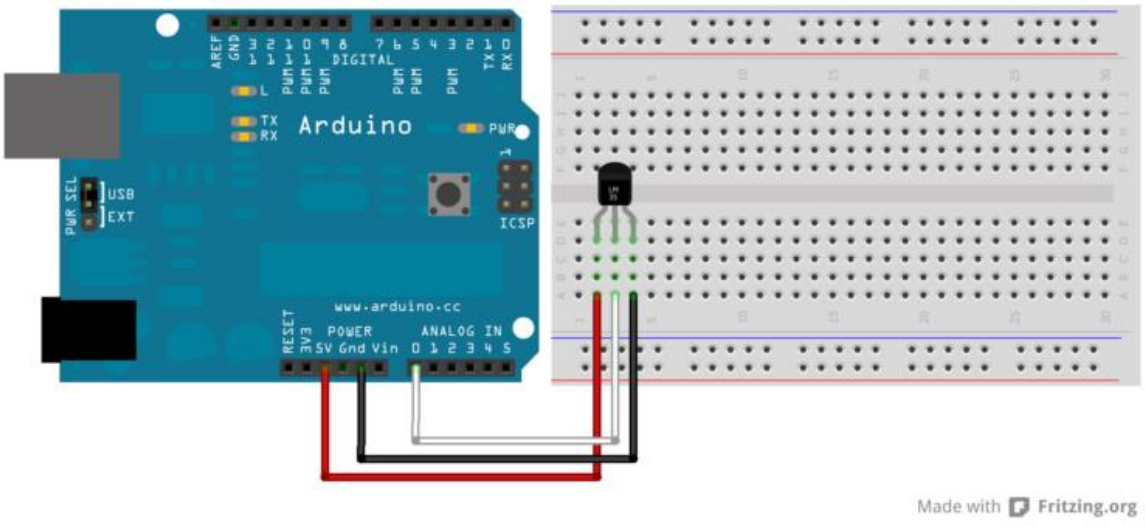

Figura 19: Montagem do projeto do LM35.

Desafio: Crie um alarme de temperatura para uma caldeira que funcione como um termômetro indicativo de leds que acende um led a cada 5°C e avisa ao operador de alguma forma que a temperatura chegou aos 55°C.

LDR (SENSOR DE LUMINOSIDADE)

O **LDR**, sigla em inglês de Light-Dependent Resistor, que significa resistor dependente de luz, nada mais é do que o que o próprio nome diz. Tipicamente, ***quanto maior a luz incidente nesse componente, menor será sua resistência.***
O LDR é constituído de um semicondutor de alta resistência, que ao receber uma grande quantidade de fótons oriundos da luz incidente, ele absorve elétrons que melhoram sua condutibilidade, reduzindo assim sua resistência. Dessa forma, esse semicondutor pode assumir resistências na ordem de mega Ohm no escuro e resistência na ordem de poucas centenas quando exposto a luz.
Neste tópico faremos um projeto que medirá e mostrará no monitor serial um valor da luminosidade do ambiente e será necessário apenas de um arduino, um sensor LDR e um resitor de 10k.

Código:

```
const int LDR = 0;
int ValorLido = 0;
void setup() {
        Serial.begin(9600);
}
void loop() {
        ValorLido = analogRead(LDR); Serial.print("Valor lido pelo LDR = "); Serial.println(ValorLido);
        delay(500); }
```

Montagem:

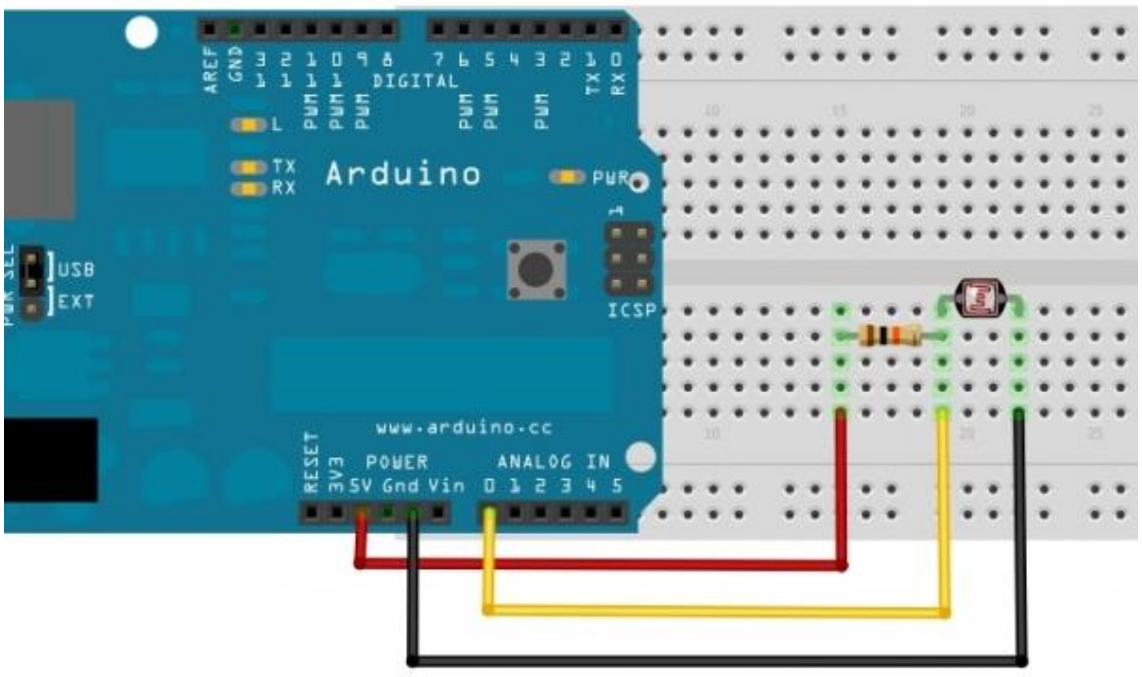

Figura 20: Montagem do projeto.

Desafio: Crie um dispositivo que ascenda um LED quando não houver mais luz no ambiente, como uma luz automática de jardim.

4. BOTÕES

Quando se trabalha com botões no arduino é necessário aplicar a lógica de “estados”, ou seja o estado o qual o botão se encontra (ligado?/Desligado?). Para que o Arduino consiga saber o estado na qual o botão se encontra devemos fazer com que cada um de seus estados defina um nível lógico diferente na entrada na qual ele está conectado. Para isso podemos utilizar o pull up interno do Arduino.

No projeto deste tópico será feito um sistema simples para operar um led com um botão. Para o led é utilizado um resistor de 300 Ω, para o botão se utiliza um resistor de 10KΩ para proteger o arduino de um curto circuito.

Código da aula:

```
const int led = 13;                 // led no pino 13
const int botao = 2;                // botao no pino 2
int estado_botao = 0;               // variável para o estado do botão

void setup() {

pinMode(led, OUTPUT);      // pino do led será saída
pinMode(botao, INPUT);     // pino do botao será entrada
}
void loop() {

estado_botao=digitalRead(2);          // A variável recebe o estado do botão
if(estado_botao == HIGH){    // Caso o botao esteja pressionado (HIGH)
digitalWrite(led,HIGH);       // Acende o LED
} else {
digitalWrite(led,LOW);        // Apaga o LED
}
}
```

Montagem:

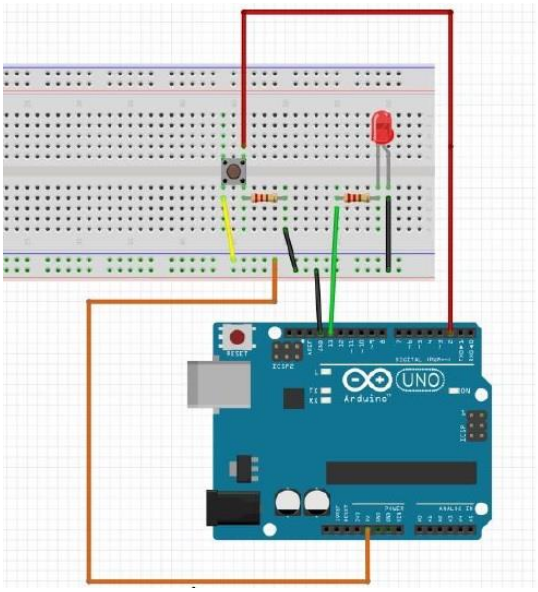

Figura 21: Montagem do projeto da aula Figura.

5. - LCD

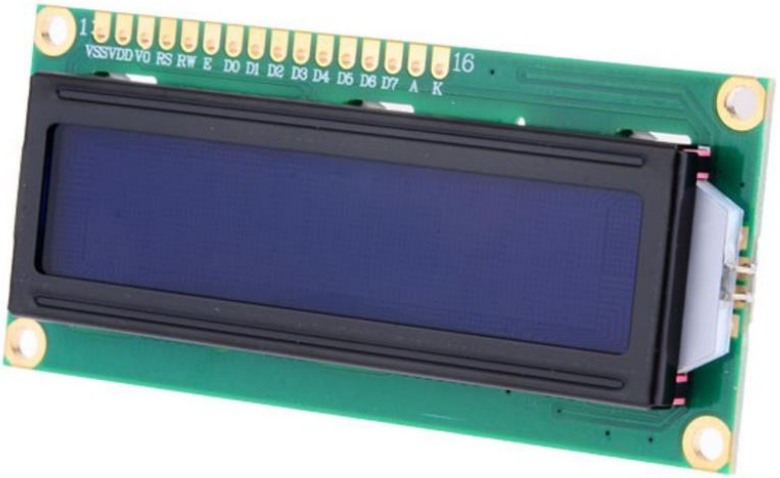

Figura 22: Exemplo LCD.

Esse display LCD tem 16 colunas e 2 linhas, com backlight (luz de fundo) azul e letras na cor branca. Para conexão, são 16 pinos, dos quais usamos 12 para uma conexão básica, já incluindo as conexões de alimentação (pinos 1 e 2), backlight (pinos 15 e 16) e contraste (pino 3).

Pino	Símbolo	Função
1	VSS	GND(Alimentação)
2	VDD	5V(Alimentação)
3	V0	Ajuste de Contraste
4	RS	Habilida/Desabilita Seletor de Registrador
5	R/W	Leitura/Escrita
6	E	Habilita Escrita no LCD
7	DB0	Dado
8	DB1	Dado
9	DB2	Dado
10	DB3	Dado
11	DB4	Dado
12	DB5	Dado
13	DB6	Dado
14	DB7	Dado
15	A	5V(Backlight)
16	K	GND(BackLight)

Figura 23: Ligação dos Pinos.

Código da aula:

```
//Carrega a biblioteca LiquidCrystal
#include <LiquidCrystal.h>

//Define os pinos que serão utilizados para ligação ao display
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
```

```
void setup()
{
  //Define o número de colunas e linhas do LCD
  lcd.begin(16, 2);
}

void loop()
{
  //Limpa a tela
  lcd.clear();
  //Posiciona o cursor na coluna 3, linha 0;
  lcd.setCursor(3, 0);
  //Envia o texto entre aspas para o LCD
  lcd.print("FILIPEFLOP");
  lcd.setCursor(3, 1);
  lcd.print(" LCD 16x2");
  delay(5000);

  //Rolagem para a esquerda
  for (int posicao = 0; posicao < 3; posicao++)
  {
    lcd.scrollDisplayLeft();
    delay(300);
}
  //Rolagem para a direita
  for (int posicao = 0; posicao < 6; posicao++)
  {
    lcd.scrollDisplayRight();
    delay(300);
  }
}
```

Montagem:

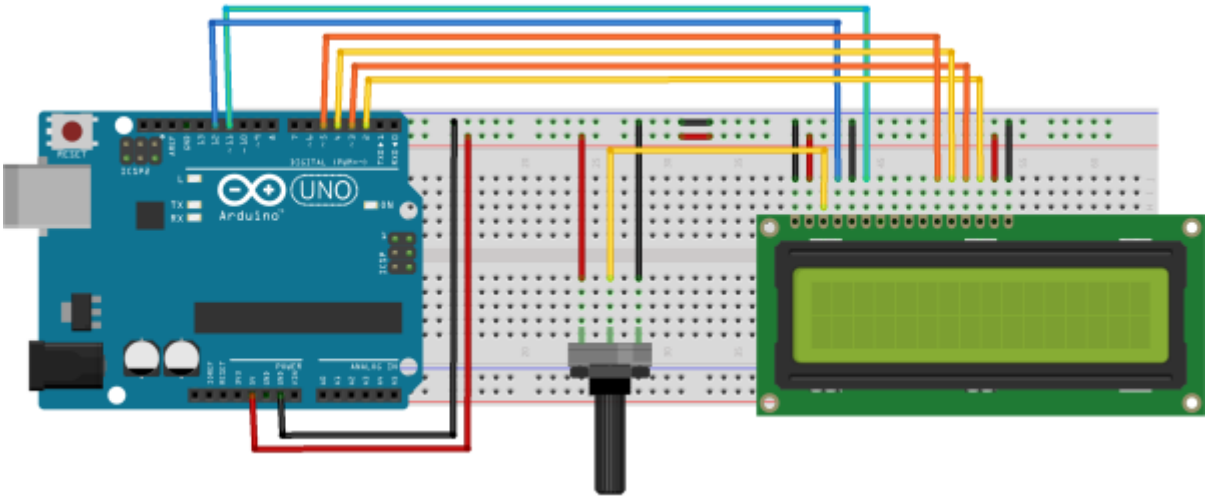

Figura 24: Montagem do projeto.

6. DISPLAY 7 SEGMENTOS

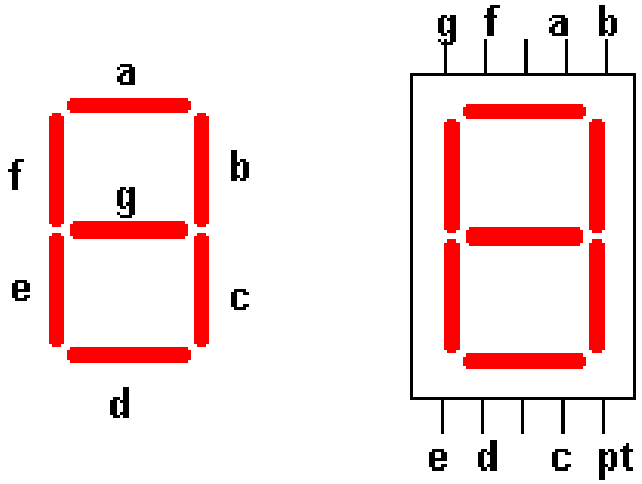

Figura 25: Funcionamento display 7 segmantos.

Código:

```
const int A = 12; // Primeiramente definimos os 7 pinos const int B = 11;
const int C = 10;
const int D = 9; const int E = 8; const int F = 6; const int G = 7;

void setup(){
pinMode(A, OUTPUT); // Todos as portas que estão os leds do display tem modo saída
pinMode(B, OUTPUT);
pinMode(C, OUTPUT);
pinMode(D, OUTPUT);
pinMode(E, OUTPUT);
pinMode(F, OUTPUT);
pinMode(G, OUTPUT);
}
void loop(){
digitalWrite(A, HIGH); //acende os leds que representam o número 0
digitalWrite(B, HIGH);
digitalWrite(C, HIGH);
digitalWrite(D, HIGH);
digitalWrite(E, HIGH);
digitalWrite(F, HIGH);
digitalWrite(G, LOW);
delay(1000); //aguarda 1 segundo para mostrar próximo número
digitalWrite(A, LOW); //acende os leds que representam o número 1
digitalWrite(B, HIGH);
 digitalWrite(C, HIGH);
digitalWrite(D, LOW);
digitalWrite(E, LOW);
 digitalWrite(F, LOW);
digitalWrite(G, LOW);
delay(1000);}
```

Montagem:

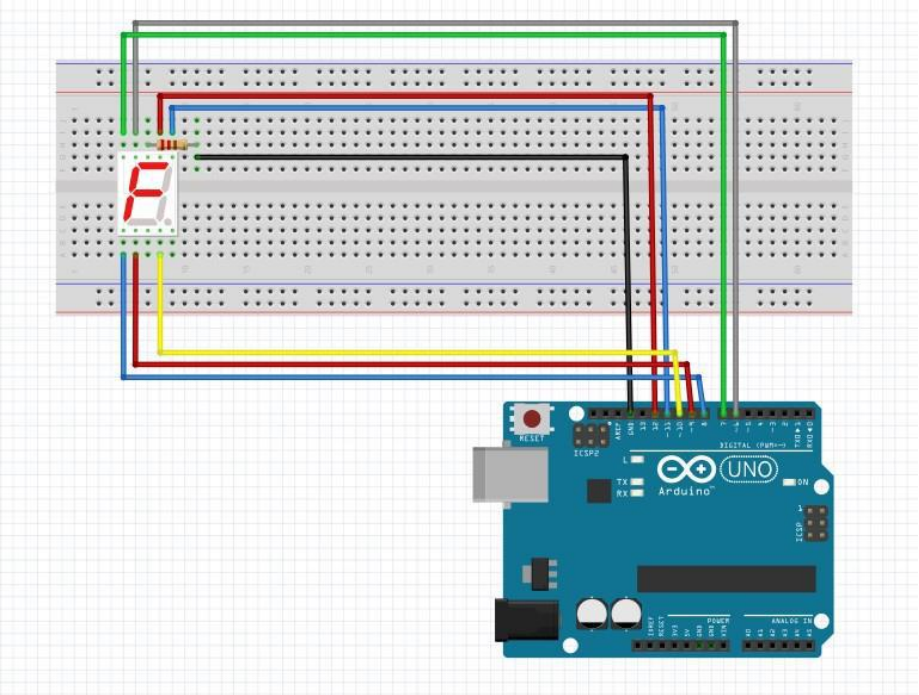

Figura 26: Montagem projeto.

PROJETO 14 – DISPLAY DE 7 SEGMENTOS COM CIRCUITO INTEGRADO

O projeto anterior foi muito trabalhoso. Para simplificar nosso trabalho, neste e em muitos outros casos, existem os chamados circuitos integrados (CI). Cada CI possui uma função diferente, sendo necessário conferir o *datasheet* de cada um em específico. O CI que utilizaremos, CI 4511, tem a função de programar o display de 7 segmentos.

Ele funciona a partir da decodificação de números binários em números decimais. Por exemplo, para os números que trabalharemos, temos:

Tabela 5: Exemplo da correspondência entre números binários e números decimais

Número em Binário	Número em Decimal
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6

0 1 1 1	7
1 0 0 0	8
1 0 0 1	9

Objetivo: Repetir o experimento anterior, utilizando um circuito integrado do tipoCódigo:

```
const int a = 4;
const int b = 5;
const int c = 6;
const int d = 7;
void setup(){
pinMode(a, OUTPUT);
 pinMode(b, OUTPUT);
pinMode(c, OUTPUT);
pinMode(d, OUTPUT);
}


void loop(){

digitalWrite(a, LOW); //Número 0
digitalWrite(b, LOW);
digitalWrite(c, LOW);
digitalWrite(d, LOW);
delay(1000);

digitalWrite(a, HIGH); //Número 1
digitalWrite(b, LOW);
digitalWrite(c, LOW);
digitalWrite(d, LOW);
delay(1000);


digitalWrite(a, LOW); //Número 2
digitalWrite(b, HIGH);
 digitalWrite(c, LOW);
digitalWrite(d, LOW); delay(1000);
}
```

Montagem:

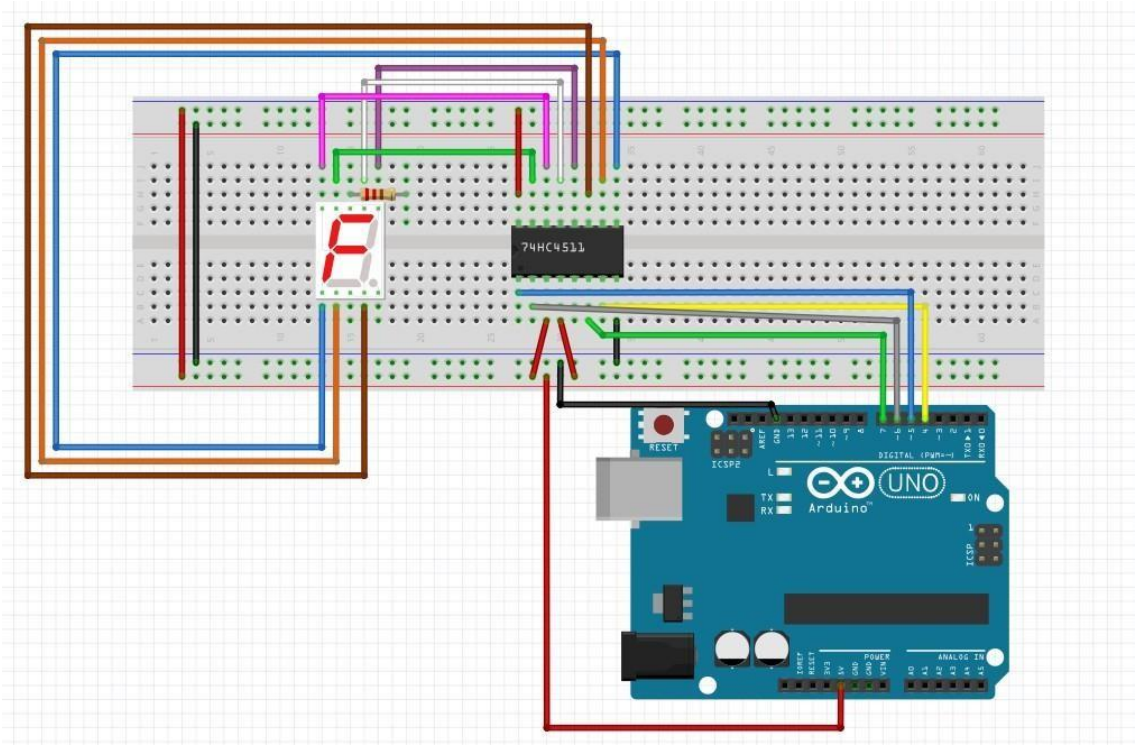

Figura 27: montagem do projeto.

6. SENSOR ULTRASSÔNICO

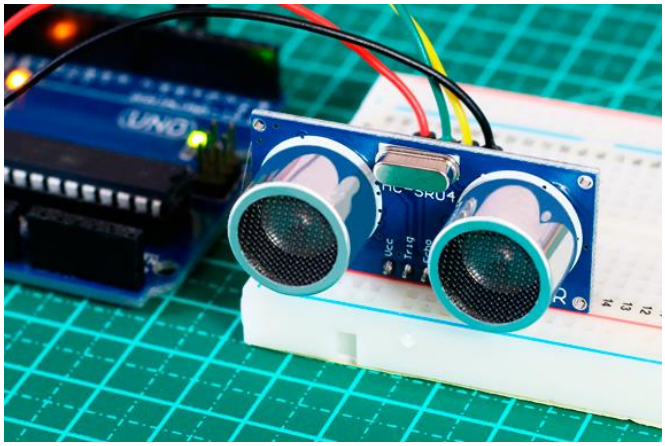

Figura 28: Sensor Ultrassônico HC-SR04.

O Sensor Ultrassônico HC-SR04 é um componente muito comum em projetos com Arduino, e permite que você faça leituras de distâncias entre 2 cm e 4 metros, com precisão de 3 mm. Pode ser utilizado simplesmente para medir a distância entre o sensor e um objeto, como para acionar portas do microcontrolador, desviar um robô de obstáculos, acionar alarmes, etc. Neste tutorial ensinaremos a conectar o HC-SR04 ao Arduino.

FUNCIONAMENTO:

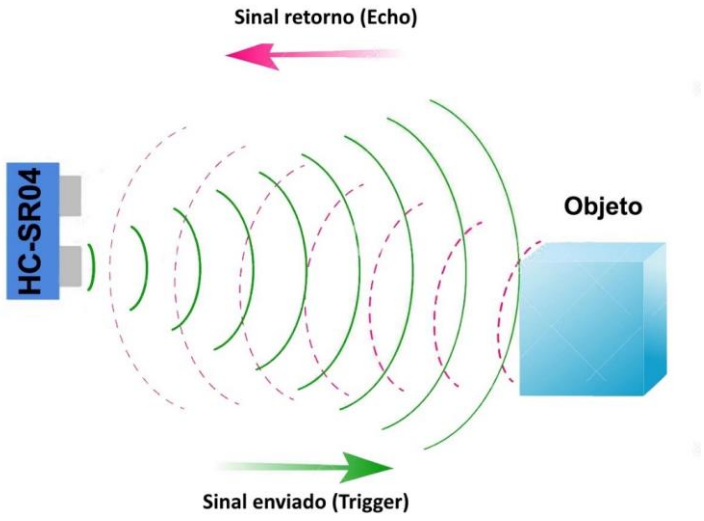

Figura 29: Funcionamento Sensor Ultrassônico.

Código da aula:

```
#include <Ultrasonic.h>
//Define os pinos para o trigger e echo
#define pino_trigger 4
#define pino_echo 5

Ultrasonic ultrasonic (pino_trigger, pino_echo);

void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.print("Distance in CM: ");
  Serial.println(ultrasonic.distanceRead());
  delay(1000);
}
```

Montagem:

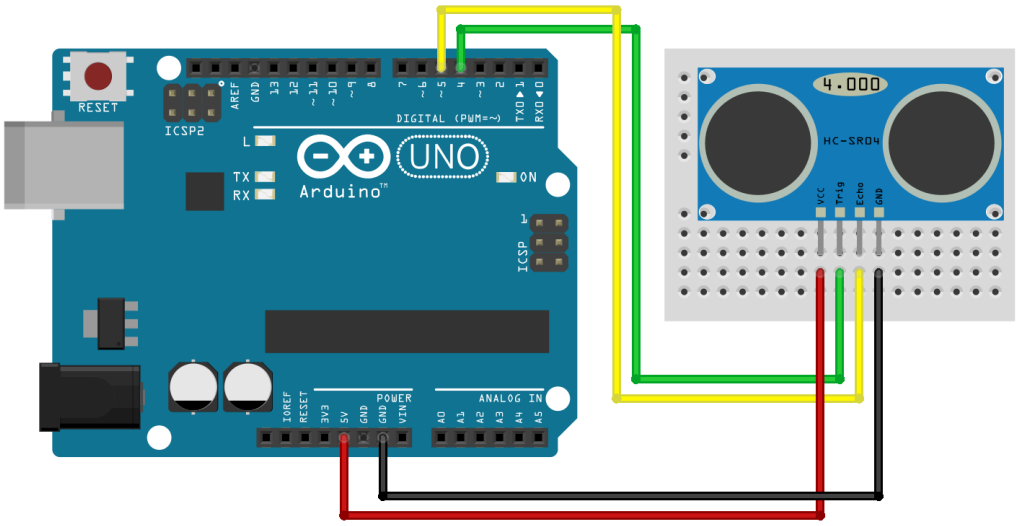

Figura 30: Sensor Ultrassônico HC-SR04.

8. AUTOMAÇÃO BÁSICA (BLUETOOTH)

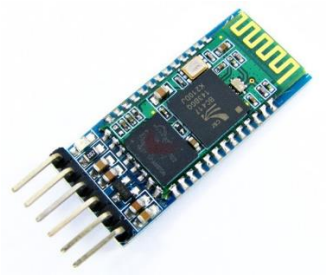

Figura 31: Módulo bluetooth.

Código da aula:

```
int readBluetooth; //Variável que irá receber o comando enviado do Android
int l1 = 3;
int l2 = 5;
int l3 = 7;

void setup() {
 Serial.begin(9600);//Inicia comunicação serial
 pinMode(l1,OUTPUT);
 pinMode(l2,OUTPUT);
 pinMode(l3,OUTPUT);
 }

void loop() {
  if(Serial.available()>0){//Verifica se algo chegou via Bluetooth
    readBluetooth=Serial.read();//Grava esse algo lido na variável
    if(readBluetooth == '5'){
      digitalWrite(l1,!digitalRead(l1));//Alterna estado da lâmpada
    }
    if(readBluetooth == '4'){
      digitalWrite(l2,!digitalRead(l2));//Alterna estado da lâmpada
    }
    if(readBluetooth == '3'){
      digitalWrite(l3,!digitalRead(l3));//Alterna estado da lâmpada
    }

  }
}
```

Montagem:

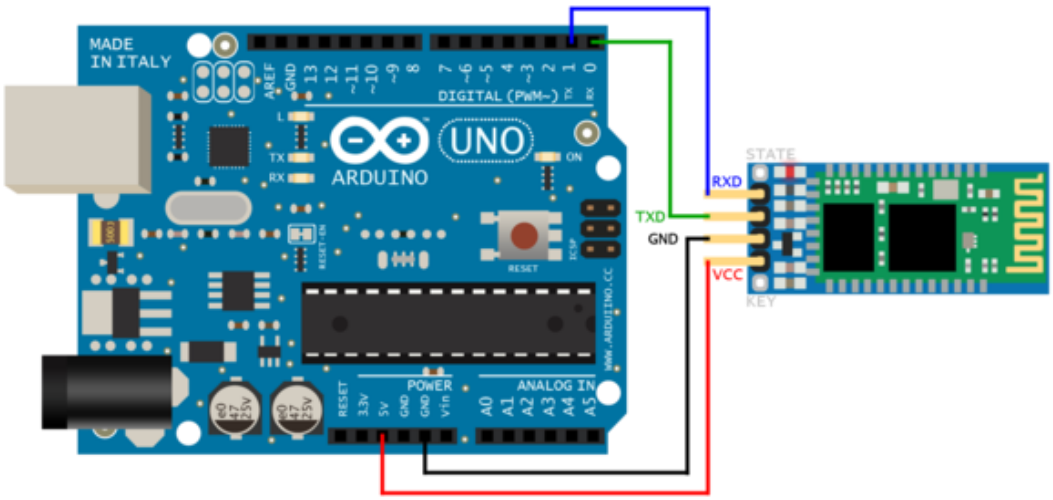

Figura 32: Montagem módulo bluetooth.

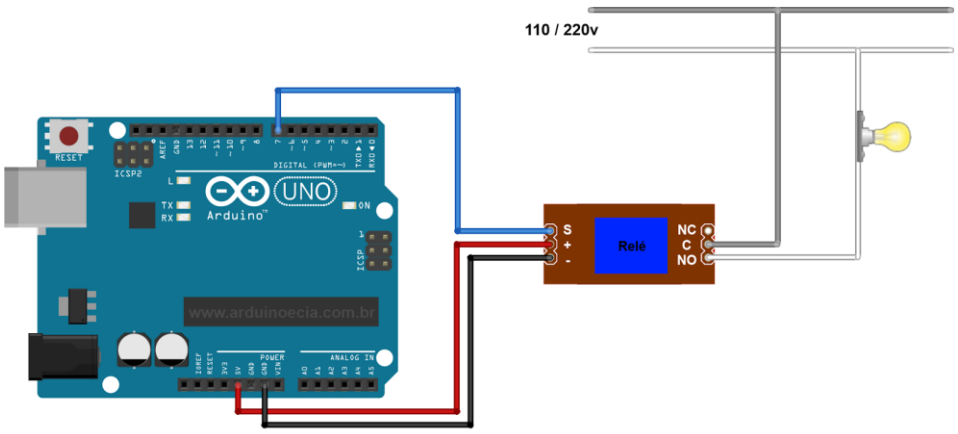

Figura 33: Montagem módulo relé.

CONCLUSÃO

A partir do que foi desenvolvido na Apostila de Arduino, permitiu-se que o discente compreenda muito bem a utilização básica de uma das plataformas mais utilizadas no mundo, que possui também uma das mais engajadas comunidades prontas para auxiliá-lo quando for necessário.

Ensinou-se o uso das ferramentas básicas para a utilização do Arduino, assim como o potencial do mesmo para projetos ainda maiores e mais complexos dos que foram apresentados na apostila. Este fator deve encorajar o discente a procurar mais utilizações diferentes a fim de aperfeiçoar-se e utilizar esse conhecimento em futuros projetos.

REFERÊNCIAS

http://www.arduinobr.com/

http://www.vidadesilicio.com.br/

https://www.filipeflop.com/blog/

http://www.comofazerascoisas.com.br/

https://www.citisystems.com.br/servo-motor/

https://www.arduino.cc/

http://www.arduinobr.com/

BIBLIOGRAFIA:

MCROBERTS, Michael. Rafael Zanolli, **Arduino Básico.** 1ª edição, São Paulo: Novatec, 2011;

ROBOCORE. **Apostila Kit Iniciante V7.2 para Arduino.** São Paulo.