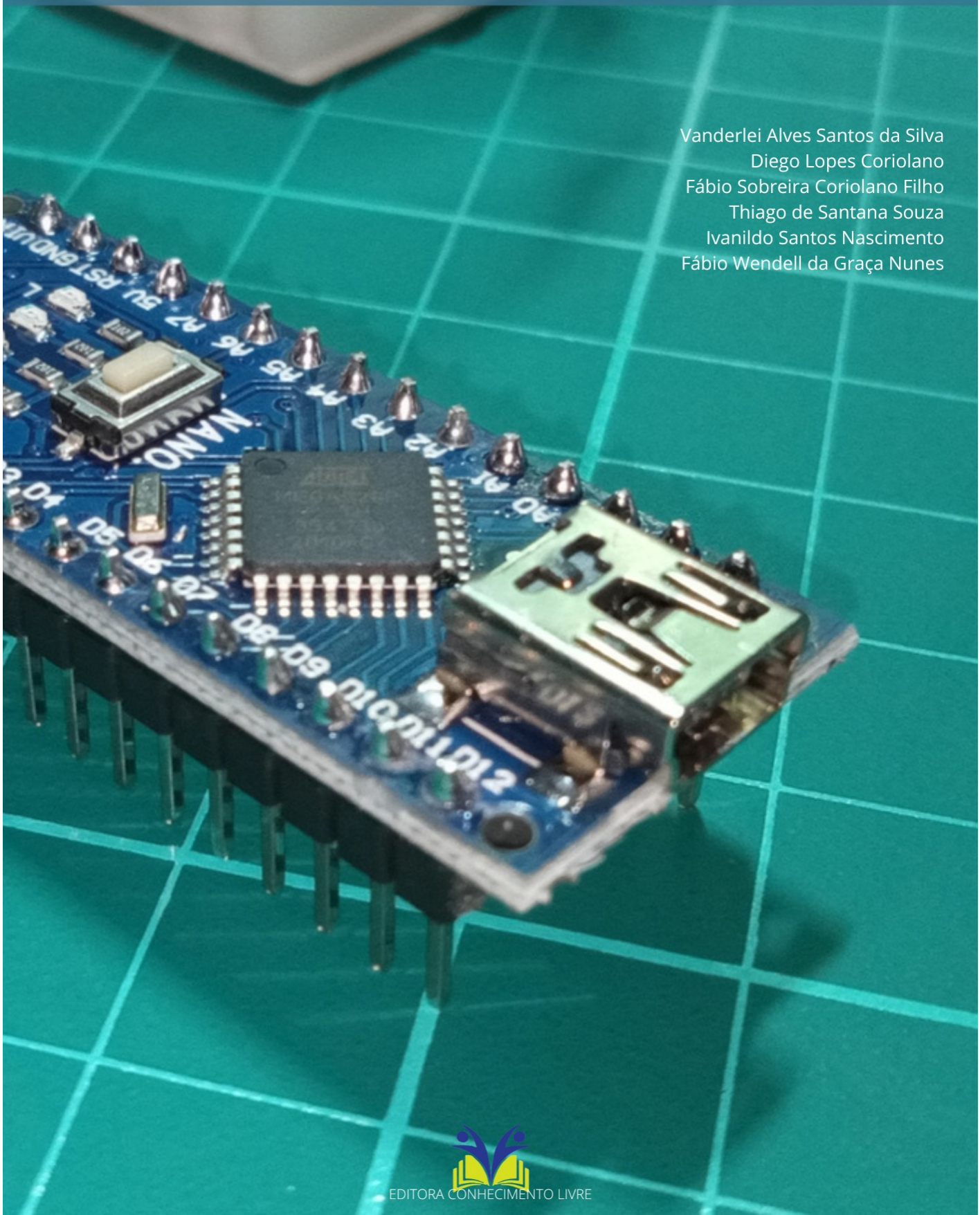
ARDUINO ESSENCIAL:
PROJETOS PRÁTICOS PARA INICIANTES



Vanderlei Alves Santos da Silva
Diego Lopes Coriolano
Fábio Sobreira Coriolano Filho
Thiago de Santana Souza
Ivanildo Santos Nascimento
Fábio Wendell da Graça Nunes

EDITORA CONHECIMENTO LIVRE

Vanderlei Alves Santos da Silva
Diego Lopes Coriolano
Fábio Sobreira Coriolano Filho
Thiago de Santana Souza
Ivanildo Santos Nascimento
Fábio Wendell da Graça Nunes

Arduino Essencial: Projetos Práticos para Iniciantes

1ª ed.

Piracanjuba-GO
Editora Conhecimento Livre
Piracanjuba-GO

1ª ed.

Dados Internacionais de Catalogação na Publicação (CIP)

Silva, Vanderlei Alves Santos da
S586A Arduino Essencial: Projetos Práticos para Iniciantes
/ Vanderlei Alves Santos da Silva. Diego Lopes Coriolano. Fábio Sobreira Coriolano Filho. Thiago de Santana Souza. Ivanildo Santos Nascimento. Fábio Wendell da Graça Nunes. – Piracanjuba-GO

Editora Conhecimento Livre, 2024

117 f.: il
DOI: 10.37423/2024.edcl956
ISBN: 978-65-5367-506-3
Modo de acesso: World Wide Web
Incluir Bibliografia

1. arduino 2. programação 3. automação I. Silva, Vanderlei Alves Santos da II. Coriolano, Diego Lopes III. Coriolano Filho, Fábio Sobreira IV. Souza, Thiago de Santana V. Nascimento, Ivanildo Santos VI. Nunes, Fábio Wendell da Graça VII. Título

CDU: 620

https://doi.org/10.37423/2024.edcl956

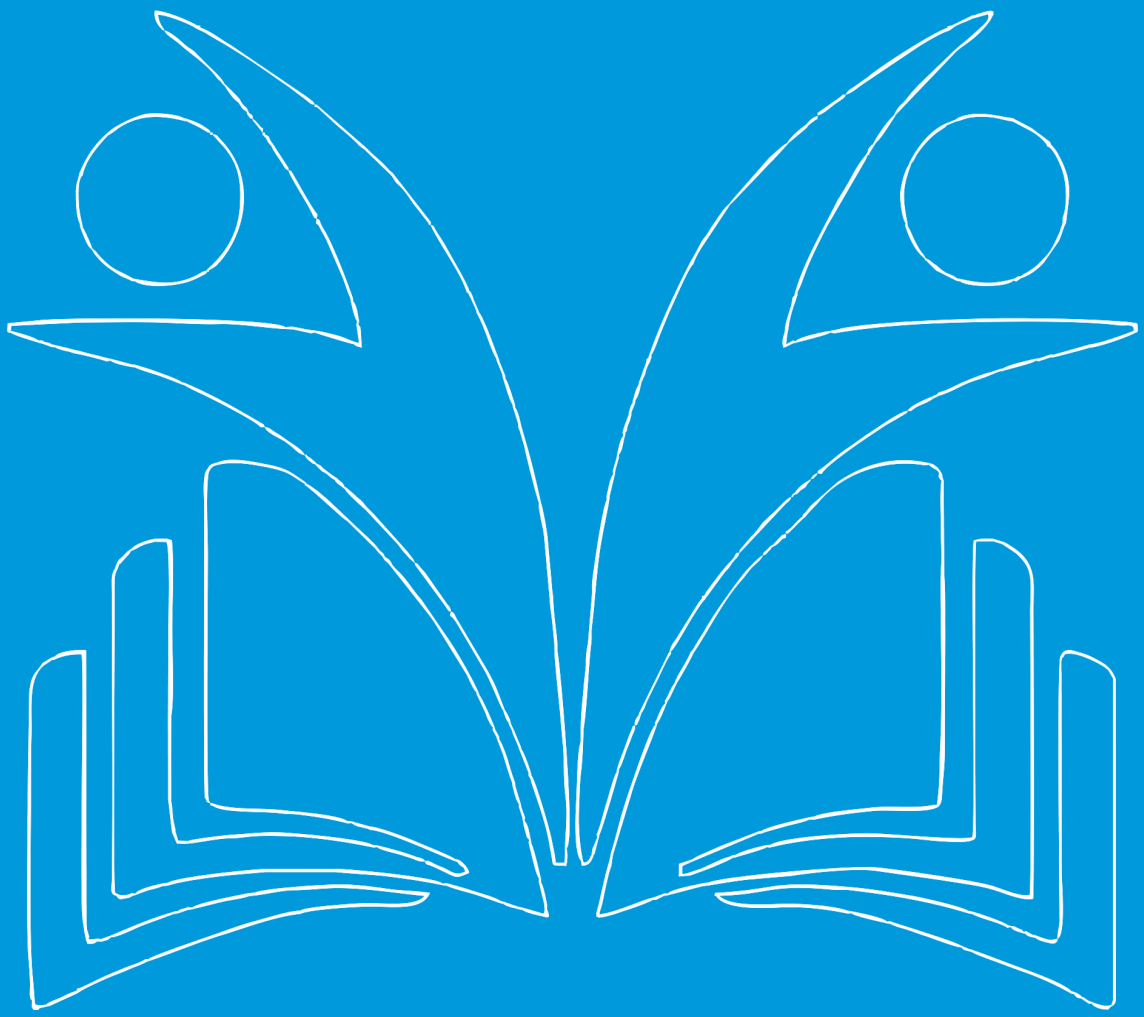
EDITORA CONHECIMENTO LIVRE

Corpo Editorial

Editora Conhecimento Livre
Piracanjuba-GO
2024

Arduino Essencial: Projetos Práticos para Iniciantes

10.37423/2024.edcl956



AGRADECIMENTOS

Gostaríamos de expressar nossa profunda gratidão à Fundação de Apoio à Pesquisa e à Inovação Tecnológica do Estado de Sergipe (FAPITEC/SE) e à Financiadora de Estudos e Projetos (FINEP) pelo apoio inestimável que nos foi concedido através do Edital FINEP/FAPITEC/SE N° 01/2020 - Programa TECNOVA II. Este projeto, que culminou na criação deste livro, só foi possível graças à visão e ao comprometimento dessas instituições com a promoção da inovação e do desenvolvimento tecnológico no Brasil.

A subvenção econômica oferecida foi fundamental para transformar nossa ideia em realidade, permitindo-nos explorar novos horizontes na educação tecnológica e na disseminação do conhecimento sobre a plataforma Arduino. A confiança depositada em nosso projeto não apenas nos motivou, mas também reforçou a importância de nosso trabalho para o avanço tecnológico e educacional.

Além do suporte financeiro, o acompanhamento técnico e o suporte contínuo da FAPITEC/SE foram cruciais para a execução bem-sucedida de nosso projeto. Agradecemos também por nos proporcionar uma plataforma para compartilhar nossos resultados com a comunidade científica e tecnológica, ajudando a inspirar futuras gerações de inventores e inovadores.

Este livro é um testemunho do que pode ser alcançado quando instituições como a FAPITEC/SE e a FINEP investem na capacidade criativa e inovadora de seus cidadãos. Estamos imensamente gratos por esta oportunidade e esperamos que nossa colaboração continue a dar frutos, contribuindo para um futuro mais inovador e sustentável.

INTRODUÇÃO

Este livro é destinado a entusiastas, estudantes, professores, e todos aqueles que se aventuram no fascinante mundo da eletrônica e da programação com Arduino. Através de uma série de projetos detalhadamente descritos, proporcionamos uma jornada didática pela programação de microcontroladores, explorando suas vastas aplicações práticas e potencializando o aprendizado de conceitos fundamentais em eletrônica.

Os projetos, cuidadosamente selecionados e apresentados neste livro, variam desde aplicações simples, como o projeto "Blink" que ensina a fazer um LED piscar, até propostas mais complexas, como o "Sequencial de LEDs" e o "Controlador de LED por Botão". Cada projeto é acompanhado de listas de materiais, esquemas de montagem detalhados e explicações passo a passo do código utilizado, garantindo que até mesmo iniciantes possam seguir sem dificuldades.

Ao integrar conceitos de eletrônica básica com programação em C++, este livro busca fornecer uma compreensão sólida dos fundamentos da automação e do controle digital, ao mesmo tempo em que estimula a inovação e a criatividade. Com isso, preparamos o leitor não só para replicar os experimentos propostos, mas também para que eles possam desenvolver suas próprias aplicações e soluções inovadoras.

Dedicamos uma atenção especial à clareza e à didática na apresentação dos projetos, de modo que o leitor possa não apenas executar, mas também entender profundamente cada aspecto envolvido no processo, desde a montagem física dos componentes até as linhas de código que fazem os circuitos ganharem vida.

Além de promover a compreensão da programação em Arduino, cada projeto também visa introduzir o leitor às práticas de resolução de problemas técnicos, desenvolvimento de pensamento crítico e habilidades de engenharia aplicada. Essas habilidades são essenciais para quem deseja avançar na área de tecnologia e inovação.

Através deste livro, esperamos não apenas educar, mas também inspirar nossos leitores a desenvolver seus próprios projetos e explorar ainda mais as possibilidades que o Arduino oferece. Encorajamos cada leitor a mergulhar nas experiências propostas, adaptá-las e expandi-las conforme sua criatividade e necessidades específicas.

Seja bem-vindo ao mundo do Arduino, onde a criatividade encontra a tecnologia, e cada projeto concluído é uma porta que se abre para novas descobertas. Vamos construir juntos um futuro mais inovador e conectado, um projeto de cada vez.

APLICAÇÕES REAIS DA PLATAFORMA ARDUINO

A plataforma Arduino, desde sua concepção, tem se destacado como uma ferramenta extraordinária para o desenvolvimento de projetos nas áreas de eletrônica e automação, tanto para entusiastas quanto para profissionais. Sua flexibilidade, facilidade de uso e baixo custo tornam o Arduino ideal para uma vasta gama de aplicações. Aqui, exploraremos alguns casos reais fascinantes que demonstram o potencial do Arduino em resolver problemas práticos e em impulsionar inovações tecnológicas.

1. Monitoramento Ambiental Em um mundo cada vez mais preocupado com questões ambientais, o Arduino tem sido utilizado para desenvolver sistemas de monitoramento ambiental acessíveis. Um exemplo notável é o uso do Arduino para monitorar a qualidade do ar em áreas urbanas. Sensores conectados a um Arduino podem coletar dados sobre poluentes, como monóxido de carbono e partículas suspensas, permitindo a análise em tempo real das condições ambientais. Esses dados podem ser usados por organizações governamentais e não governamentais para implementar estratégias mais eficazes na melhoria da qualidade do ar.

2. Automação Residencial O uso do Arduino na automação residencial tem transformado a maneira como interagimos com nossos lares. Projetos baseados em Arduino podem controlar luzes, sistemas de segurança, termostatos e outros aparelhos domésticos, promovendo não apenas conforto mas também eficiência energética. Por exemplo, um sistema baseado em Arduino pode ser programado para ajustar automaticamente a iluminação e a temperatura de uma casa baseado nos hábitos dos moradores, resultando em significativa economia de energia.

3. Agricultura Inteligente Na agricultura, o Arduino ajuda a implementar técnicas de agricultura de precisão, que aumentam a eficiência e a produtividade das culturas. Sistemas de irrigação inteligentes baseados em Arduino, por exemplo, utilizam sensores de umidade do solo para automatizar a irrigação, garantindo que as plantas recebam a quantidade exata de água necessária. Isso não apenas economiza água, mas também maximiza o crescimento das plantas.

4. Projetos de Saúde e Bem-estar O Arduino também encontrou aplicações significativas na área de saúde. Um exemplo é a criação de dispositivos de assistência para pessoas com deficiência. Projetos

DIY (faça você mesmo) incluem luvas que convertem a linguagem de sinais em texto e próteses de baixo custo que podem ser personalizadas para as necessidades individuais dos usuários. Esses dispositivos não apenas melhoram a qualidade de vida, mas também são financeiramente acessíveis.

5. Robótica Educacional No campo educacional, o Arduino serve como uma ferramenta essencial no ensino de programação e robótica para estudantes. Robôs controlados por Arduino, que podem seguir linhas ou resolver labirintos, oferecem aos alunos uma maneira prática e divertida de aprender conceitos de engenharia, matemática e ciências da computação.

6. Arte e Design Artistas e designers utilizam o Arduino para explorar novas formas de expressão artística. Instalações interativas que respondem ao ambiente ou ao público, por exemplo, usam sensores conectados a um Arduino para alterar luzes, sons e movimentos com base na interação do espectador, criando experiências imersivas e dinâmicas.

VANTAGENS E DESVANTAGENS DA UTILIZAÇÃO DA PLATAFORMA ARDUINO

A acessibilidade é, sem dúvida, uma das principais vantagens do Arduino. Com seu custo relativamente baixo e a disponibilidade generalizada de seus componentes, o Arduino torna a tecnologia acessível a um público amplo, incluindo estudantes, *hobbistas* e startups. Esta acessibilidade financeira incentiva a experimentação e inovação sem a necessidade de um grande investimento inicial, democratizando o acesso à aprendizagem de programação e eletrônica.

Outro ponto forte do Arduino é a facilidade de uso. Projetado tanto para principiantes quanto para usuários avançados, o Arduino oferece uma plataforma de desenvolvimento integrado (IDE) que simplifica significativamente o processo de programação e upload de códigos para a placa. Isso torna o aprendizado de conceitos fundamentais de programação e eletrônica intuitivo e acessível, eliminando muitas barreiras que geralmente desencorajam os novatos.

A vasta comunidade de usuários do Arduino é um recurso inestimável. A popularidade do Arduino resultou em uma extensa comunidade online que oferece suporte, tutoriais, projetos de exemplo e bibliotecas. Isso facilita a resolução de problemas e a colaboração em projetos, além de ser uma fonte constante de inspiração para novas ideias e aplicações.

Em termos de compatibilidade e modularidade, o Arduino se destaca. A plataforma é compatível com uma ampla variedade de módulos e sensores, o que permite aos usuários construir projetos complexos e multifuncionais. Esta modularidade facilita a experimentação e expansão dos projetos, permitindo uma personalização sem precedentes e adaptabilidade.

Por fim, o modelo de código aberto do Arduino promove uma cultura de inovação e colaboração. Tanto o hardware quanto o software do Arduino são de código aberto, permitindo aos usuários modificar e melhorar seus sistemas como necessário. Isso não só incentiva a personalização e o aprimoramento contínuo, mas também ajuda a comunidade a crescer e a evoluir juntos.

Apesar das muitas vantagens, o Arduino tem suas limitações. Em termos de desempenho, a plataforma pode não ser adequada para projetos que exigem processamento intensivo ou capacidades gráficas avançadas. Sua memória limitada e velocidade de processamento podem ser insuficientes para aplicações mais exigentes, o que pode frustrar usuários que buscam desenvolver projetos com requisitos de alta performance.

A segurança e a robustez do Arduino também podem ser preocupantes. As placas Arduino não são projetadas para ambientes industriais ou aplicações onde a falha não é uma opção. A falta de mecanismos de segurança robustos faz com que não seja ideal para projetos que exigem alta confiabilidade e segurança, como sistemas de controle críticos ou dispositivos médicos.

Profissionalmente, o Arduino pode servir como um trampolim, mas pode não ser suficiente para aqueles que precisam de um conhecimento mais profundo de eletrônica e programação. Profissionais podem encontrar-se necessitando migrar para outras plataformas mais potentes e especializadas para atender às demandas específicas do mercado.

Por último, há uma percepção de que o Arduino é menos sério ou capaz do que outras plataformas de microcontroladores. Essa percepção pode afetar negativamente a escolha do Arduino para projetos comerciais sérios, onde a confiabilidade e a imagem profissional são cruciais.

Considerando todas essas facetas, o Arduino oferece uma plataforma de entrada excepcional para explorar o mundo da eletrônica e da programação. No entanto, é crucial reconhecer suas limitações ao planejar projetos que vão além da prototipagem e dos experimentos educacionais.

BLINK

OBJETIVO

Esse projeto apresentará a forma de configurar um pino do Arduino como saída digital com a finalidade de fazer um LED piscar, além de conhecermos a maneira mais simples de se adquirir um atraso no funcionamento de nosso dispositivo.

MONTAGEM DO CIRCUITO

A figura abaixo mostra como devem ser feitas as ligações dos componentes ao Arduino com o uso de uma protoboard. Preste bastante atenção em cada ligação.

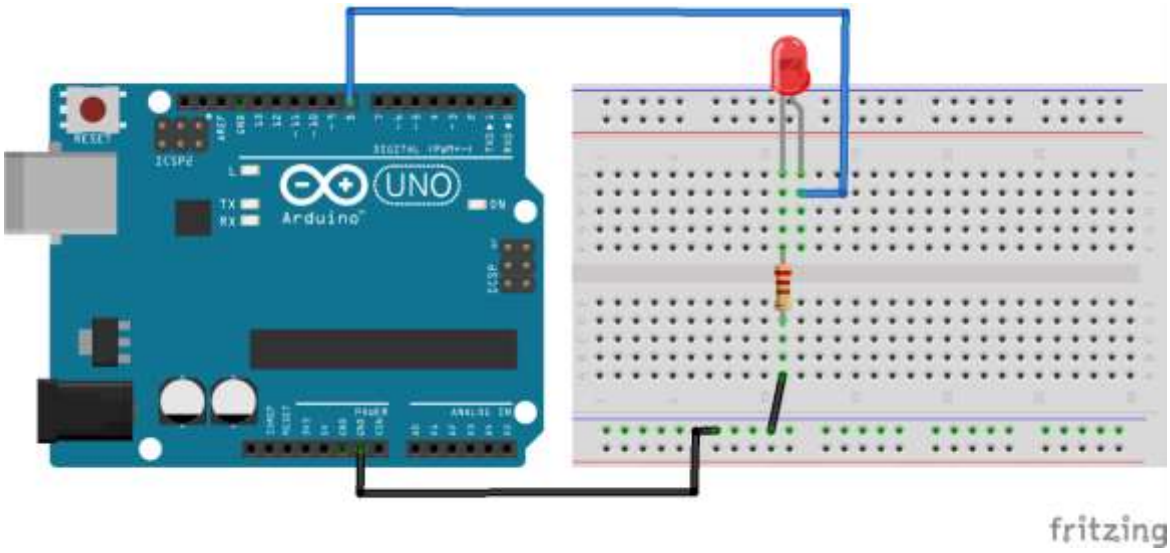


Figura 1 - *Montagem do circuito Blink. Ligação do LED ao pino 8.*

Lista de materiais

Arduino UNO
1 LED vermelho
1 Resistor 220Ω (vermelho, vermelho, marrom)
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

Segue abaixo o sketch que deverá ser escrito na Arduino IDE para programar a placa.

```
/************************
 * ------ Blink ------ *
 ************************/

void setup() {

  pinMode(8, OUTPUT);

}

void loop() {

  digitalWrite(8, 1);
  delay(1000);
  digitalWrite(8, 0);
  delay(1000);

}
```

EXPLICANDO O SKETCH

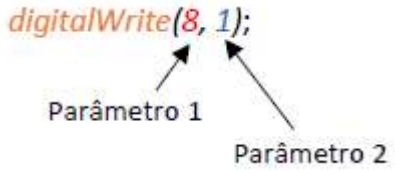
Vamos começar explicando sobre a função presente na linha 5, a função *setup()*. Esta função tem a finalidade de executar tarefas e configurações logo após o Arduino ser alimentado e cada tarefa e/ou configuração serão realizados apenas uma vez, mesmo que o circuito permaneça energizado. Sendo assim, podemos notar a presença de uma função dentro de *setup()*:

A seguinte função recebe dois parâmetros que estão separados por vírgula, no primeiro colocamos o número correspondente ao pino do Arduino que queremos programar, que no caso foi o pino 8, e no segundo colocamos o comando que define se o pino será saída ou entrada de dados digitais. Em nosso projeto usamos, como segundo parâmetro o comando **OUTPUT**, o que significa que estamos fazendo com que o pino 8 se torne uma saída digital. Caso colocássemos **INPUT** no lugar de **OUTPUT** estaríamos determinando esse pino como uma entrada digital.

Resumindo: A função ***pinMode*** é usada para determinar se um pino será entrada ou saída digital e deverá ser executada apenas uma única vez quando o Arduino for energizado, por isso foi escrita dentro da função *setup()*.

Na linha 11 temos a função *loop()*, a qual serve para fazer o Arduino permanecer executando a programação pretendida, que nesse caso é piscar um LED, durante o tempo em que a placa estiver sendo alimentada. Portanto, quando o programa entra na função *loop()*, não sai mais dela e executa as funções que estiverem entre suas chaves repetidamente sem parar.

Na linha 13 podemos ver a função que possui a capacidade de enviar nível lógico baixo ou alto para uma saída digital. Observe:



Esse é uma função que também deve receber dois parâmetros. No primeiro parâmetro colocamos o número do pino que foi definido como saída digital, que em nosso caso foi o pino 8, e no segundo parâmetro colocamos o comando que envia nível lógico alto ou baixo. Em nosso projeto estamos

enviando nível lógico alto, representado pelo número 1, na linha 13 do sketch. No segundo parâmetro, no lugar do número 1 podemos usar o comando HIGH para realizar a mesma tarefa, ou seja, enviar nível lógico alto e com isso fazer o LED acender.

Resumindo: Na linha 13 fizemos o LED acender com o uso da função descrita.

Para fazer o LED permanecer aceso por um certo tempo, na linha 14 usamos a função *delay(1000)*.

A função ***delay*** recebe apenas um parâmetro, um número inteiro que representa o tempo em milissegundos, que em nosso sketch é de 1 segundo, por isso o valor **1000** milissegundos. Podemos alterar esse número para qualquer valor maior que zero para obtermos o tempo em milissegundos.

Na linha 15 voltamos com a função ***digitalWrite***, mas observe que agora, no segundo parâmetro, temos o valor 0 (zero), comando este que envia nível lógico baixo através do pino 8, para fazer o LED ficar apagado. No lugar do zero podemos usar o comando LOW, o qual realiza a mesma tarefa de apagar o LED.

Novamente na linha 16 usamos a função *delay* para pausar o programa por mais 1000 milissegundos (1 segundo).

Resumindo: Na linha 13 fizemos o LED acender; na linha 14 pausamos o programa por 1 segundo, o que manteve o LED aceso por esse tempo; na linha 15 fizemos o LED apagar e na linha 16 novamente pausamos o programa por mais 1 segundo.

Como todas as funções entre as linhas 13 e 16 estão dentro da função *loop()*, após terminar de executar a última tarefa presente na linha 16, o programa volta automaticamente para a linha 13 fazendo tudo de novo infinitamente.

FUNCIONAMENTO

Ao ligar o Arduino, o LED (LD1), ligado ao pino 8 escolhido no momento da programação, irá piscar com intervalos de tempos de 1 segundo, determinado pela função *delay*. Como o código foi escrito dentro da função *loop()*, o LED permanecerá piscando enquanto o circuito estiver recebendo alimentação.

O resistor R1, ligado entre o GND do Arduino e o terminal Cátodo (-) do LED teve seu valor escolhido conforme a fórmula:

$$R1 = \frac{Vcc - V_{led}}{I_{led}}$$

Onde:

Vcc=$Tensão\ do\ nível\ alto\ em\ volts$ ($no\ Arduino$ é de $5V$);

V_{led}=$Tensão\ de\ alimentação\ do\ LED$ ($Para\ o\ LED\ vermelho\ temos$ $1{,}8V$);

I_{led}=$Corrente\ elétrica\ para\ o\ LED$ ($para\ a\ maioria\ dos\ LEDs\ de\ baixa\ potência$ é de $0{,}02A$).

Logo:

$$R1 = \frac{5 - 1{,}8}{0{,}02}$$

$$R1 = \frac{3{,}2}{0{,}02}$$

$$R1 = 160\ Ohms$$

Portanto, o menor valor para R1 seria de 160 Ohms, mas usamos um resistor com resistência de 220 Ohms por ser fácil de conseguir no mercado e por não afetar o funcionamento do circuito e podendo garantir maior vida útil para o LED.

BLINK COM CONTROLE DE OSCILAÇÃO

OBJETIVO

Com o projeto proposto, será possível compreender a possibilidade de exercer controle sobre o funcionamento de um dispositivo via programação. Também será observada a atuação da função *setup()*, o que mostrará que esta função realiza uma determinada tarefa apenas uma vez, ou seja, somente no momento em que o Arduino é ligado. Dois elementos importantes que serão vistos nesse projeto são o uso de uma diretiva de compilação e variáveis, as quais são de grande importância em todas as linguagens de programação.

MONTAGEM DO CIRCUITO

O esquema de ligações para montagem desse circuito é apresentado na figura abaixo.

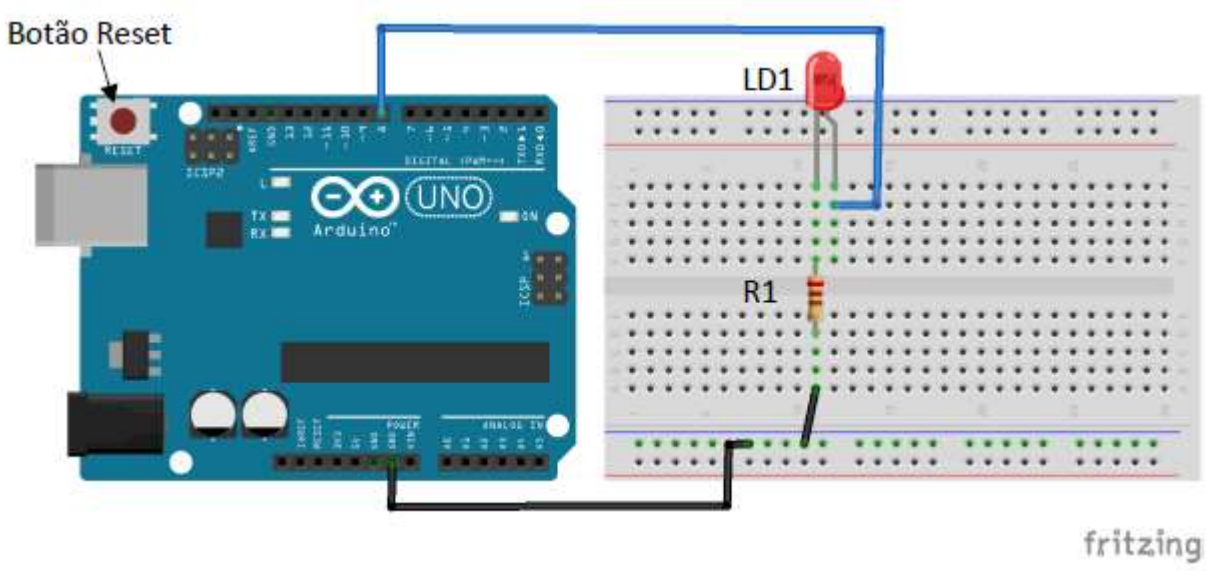


Figura 2 - *Montagem do circuito eletrônico.*

LISTA DE MATERIAIS

Arduino UNO
1 LED vermelho
1 Resistor 220Ω (vermelho, vermelho, marrom)
Protoboard
Fios ou cabinhos jumpers
Cabo USB

PROGRAMAÇÃO

Para realizar a programação desse circuito, escreva o seguinte sketch no Arduino IDE:

```
1  /*********************************
2   * ------ Blink Controlado ------ *
3   *********************************/
4
5  #define LED 8
6
7  int tempo = 500;
8  int quant = 5;
9
10 void setup() {
11
12   pinMode(LED, OUTPUT);
13
14   for(int i = 0; i < quant; i++)
15   {
16     digitalWrite(LED, HIGH);
17     delay(tempo);
18     digitalWrite(LED, LOW);
19     delay(tempo);
20   }
21
22 }
23
24 void loop() {
25
26
27
28 }
```

EXPLICANDO O SKETCH

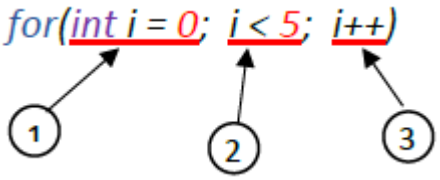
O sketch de programação desse projeto nos traz algo interessante, uma diretiva de compilação, a qual pode ser notada na linha 5. Uma diretiva de compilação é um comando que será executado no momento da compilação1 do código e a diretiva usada aqui é a *#define*, ela tem a capacidade de dar nome a um objeto, que em nosso caso, renomeamos o pino 8 para *LED*, ou seja, para a programação, a palavra *LED* representará o pino 8. Esse recurso facilita muito a vida do programador deixando o código mais amigável, além de não consumir espaço em memória para renomear um pino.

Nas linhas 7 e 8, temos duas declarações de variáveis globais, chamadas de *tempo e quant*, e foram definidas como sendo do tipo *int*, o que significa que elas só podem armazenar números inteiros. Note a presença do sinal de igualdade na frente de cada uma delas, esse sinal, nessas condições, é chamado de ***recebe***, portando, o que temos na linha 7 é lido da seguinte forma*: tempo* ***recebe*** *500*; da mesma forma como na linha 8 devemos ler *quant* ***recebe*** *5*. O uso de uma variável é semelhante ao da diretiva de compilação apresentada, com a diferença de que elas consomem espaço em memória, no entanto, se limita a um determinado tipo de dados. Existem diversos tipos de variáveis além do tipo *int*, mas serão vistas no decorrer de novos projetos.

Uma variável pode ter seu valor trocado quantas vezes forem necessárias, por isso possuem esse nome, e se tornam ainda mais importante quando um determinado valor deverá aparecer em diversos pontos diferentes no código. Imagine o seguinte: Suponha que o valor 500 deva ser escrito em várias outras linhas de um código que esteja elaborando e depois de muitas linhas, descobre-se que o valor 500 não é mais adequado e precisará ser trocado. O grande problema aqui seria ter que sair procurando onde foram digitados todos os números 500 para substituí-los por outro valor. Com o uso de uma variável, esse problema é eliminado, uma vez que declaramos uma variável com um certo valor, bastaria usar o nome dela em todas as linhas que forem necessárias e se, por um acaso, surgir a necessidade de trocar o valor, basta mudar o número que foi atribuído em sua declaração e com isso todas as outras linhas que possuem tal variável serão atualizadas. Ou seja, a variável representará o número ou informação que ela recebe.

Após as declarações das variáveis, entramos na função *setup()*. Na linha 12, nos deparamos com a função ***pinMode***, a qual está fazendo o pino 8, que agora se chama ***LED***, ser uma saída de dados através do **comando *OUTPUT*.**

Na linha 14, encontramos uma das quatro funções fundamentais em qualquer linguagem de programação, a função *for* usada para repetição e contagens. Vamos entender como ela funciona:



Onde:

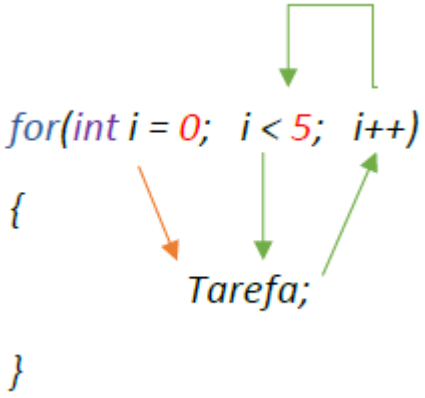
1 – Declaração de variável local, que aqui chamamos de i, mas pode ser dado o nome que quiser. Essa variável só funcionará para o *for*, não podendo usar seu valor fora dessa função por ter sua declaração feita localmente. No exemplo, essa variável i está recebendo inicialmente o valor zero, o que significa que a contagem que o *for* irá fazer começará a partir desse valor.

2 – Condição que mantém a função *for* funcionando, quando essa condição não estiver sendo mais satisfeita, o *for* para de ser executado e a programação sai dele.

3 – Incremento da variável i. A cada realização de uma tarefa, a variável i vai tendo o seu valor atual somado com mais **1**, com isso vai adquirindo cada vez um valor a mais gradativamente. A inscrição i++ é equivalente a i = i + **1**, assim, o valor atual em i é somado com mais 1.

Com essas informações, podemos entender que enquanto i for menor que 5, o que estiver entre as chaves será executado; após a execução, a variável i é incrementada e depois analisada para ver se ainda atende a condição, ou seja, verifica se i ainda é menor que 5, se sim realizará novamente o que estiver para ser executado entre as chaves. Esse processo se repete até que i chegue ao valor 5.

Veja abaixo um esquema que representa o funcionamento da função *for*, também chamada de *laço for*. Observe que após a declaração da variável local i, o programa executa a tarefa e depois fica circulando entre tarefa, incremento e análise da condição. A declaração só é realizada no início.



Em nosso projeto, no lugar do número **5** temos a variável *quant*, a qual foi inicializada em sua declaração com o valor **5**, portanto, ela representa este número. Caso seu valor seja trocado por outro, ela assumirá esse novo valor e o ***laço for*** contará até esse novo valor.

Na linha 16, já dentro do *laço for*, temos a função ***digitalWrite*** enviando nível lógico alto para o ***LED*** através do comando ***HIGH***. Isso faz o ***LED*** acender e assim permanecer.

Na linha 17, encontramos a função de retardo ***delay***, e como parâmetro ela está recebendo a variável *tempo*, a qual vimos no início que está recebendo o valor 500, com isso temos um retardo de 500 milissegundos na programação (mesmo que ***delay****(500)*), o que indica que o programa ficará parado sem fazer mais nada por esse tempo. Após o final desse tempo, o programa passará para a linha seguinte.

Na linha 18, a função ***digitalWrite*** volta, mas agora estará enviando nível lógico baixo para o *LED* por meio do comando ***LOW***, fazendo o **LED** apagar e assim permanecer.

Por último, na linha 19, é possível notar novamente a presença da função ***delay*** com o mesmo propósito já explicado linhas atrás.

FUNCIONAMENTO

Quando o Arduino é energizado, tudo o que está na função *setup()* será executado, com isso a função **for** começa sua contagem fazendo o LED piscar na quantidade de vezes e intervalos de tempo determinados na declaração das variáveis.

```
7  int tempo = 500;
8  int quant = 5;
```

Alterando o valor da variável *tempo* para 1000 e enviando novamente o código para o Arduino, perceberá que o LED piscará mais lentamente, com intervalos de 1 segundo. Se alterar a variável *quant* para 10 e enviar novamente o código para a placa, irá notar que agora o LED piscará 10 vezes com intervalos de 1 segundo. Vá fazendo experiências com novos valores.

Observe que o LED piscará somente a quantidade de vezes para a qual foi programado e depois permanecerá apagado durante todo o tempo em que o Arduino esteja ligado. Para ver ele funcionar novamente é preciso pressionar o botão Reset na placa ou desligar o cabo USB e ligar novamente. Isso ocorre porque fizemos nossa programação dentro da função *setup()*, o que prova que essa função só realiza as tarefas no momento em que o Arduino for ligado e depois não faz mais nada, passando a programação para a função *loop()*, que em nosso código está vazia.

GIROFLEX DE CARRO POLICIAL

OBJETIVO

Neste projeto, veremos como acionar duas saídas digitais de forma intermitente, fazendo com que dois LEDs de cores diferentes pisquem alternadamente, simulando luzes de carro de polícia. Aqui apenas reforçaremos os conhecimentos obtidos com o projeto Blink, explorando o uso da diretiva *#define* e estudando a possibilidade de acionar mais dispositivos através de portas digitais (pinos digitais).

MONTAGEM DO CIRCUITO

Observe atentamente as ligações mostrada na figura abaixo para realizar a montagem do circuito.

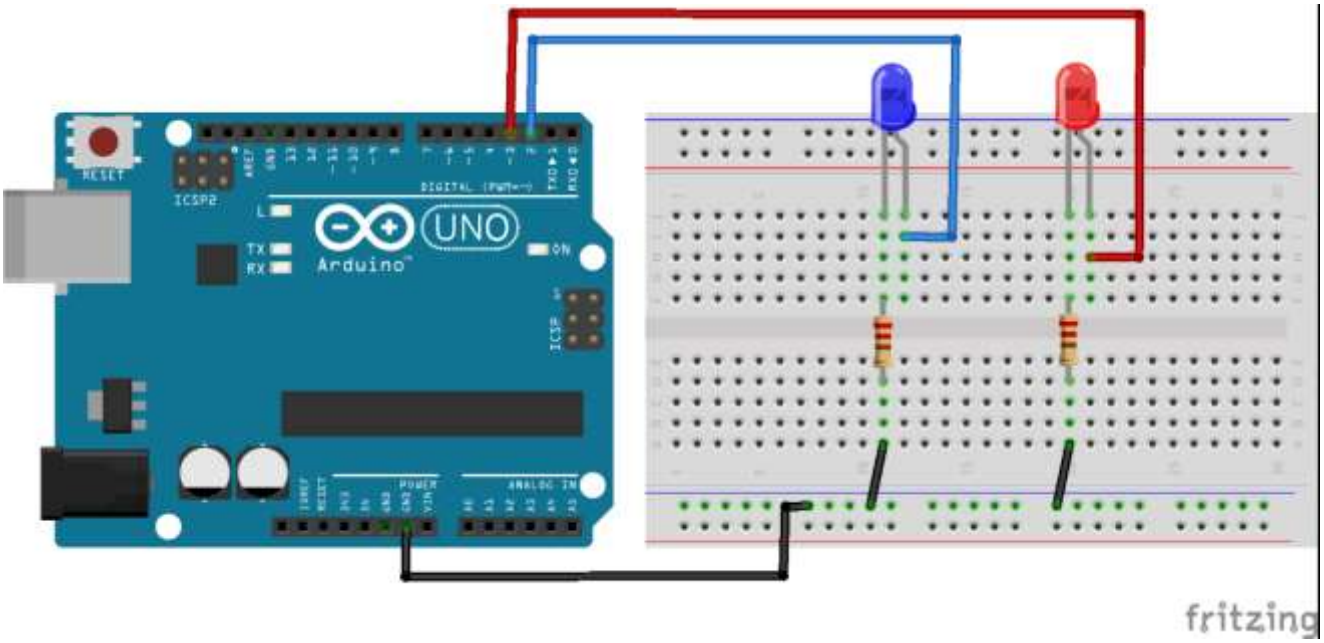


Figura 3 *- Montagem do circuito.*

Lista de materiais

Arduino UNO
LD1 - LED azul
LD2 – LED vermelho
R1 e R2 - Resistor 220Ω (vermelho, vermelho, marrom)
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

O sketch seguinte traz a programação, também chamada de código, do circuito proposto. Digite no Arduino IDE tomando o cuidado de não esquecer nenhuma pontuação e obedecendo a escrita no que diz respeito às letras maiúsculas e minúsculas, esses detalhes escritos de maneira errada podem trazer mal ou nenhuma funcionalidade ao circuito.

```
1  /**********************************
2   * ----- Giroflex Policial ----- *
3   **********************************/
4
5  #define AZUL 2
6  #define VERMELHO 3
7
8  void setup() {
9
10   pinMode(AZUL, OUTPUT);
11   pinMode(VERMELHO, OUTPUT);
12
13 }
14
15 void loop() {
16
17   digitalWrite(AZUL, HIGH);
18   digitalWrite(VERMELHO, LOW);
19   delay(500);
20
21   digitalWrite(AZUL, LOW);
22   digitalWrite(VERMELHO, HIGH);
23   delay(500);
24
25 }
```

EXPLICANDO O SKETCH

Inicialmente, nas linhas 5 e 6, notamos a presença da diretiva de compilação *#define*, aqui ela está renomeando o pino 2 como ***AZUL*** e o pino 3 como ***VERMELHO***. Por boa prática de programação, é recomendado escrever os nomes definidos em letras maiúsculas, para que assim possam ser facilmente separadas visualmente das variáveis quando houver. Nesse projeto não ocorreu a necessidade de usar variáveis.

Já na função *setup()*, nas linhas 10 e 11, a função ***pinMode*** está fazendo com que a porta 2, agora chamada de ***AZUL*** e a porta 3, denominada de ***VERMELHO***, sejam saídas digitais, isso graças ao comando ***OUTPUT***.

Dentro da função *loop()*, temos o desenvolvimento da ideia desse projeto. Na linha 17, temos a função ***digitalWrite*** enviando nível lógico alto para o pino **AZUL**, com o comando ***HIGH***, ao passo que a mesma função, na linha 18 está enviando nível lógico baixo, com o comando ***LOW,*** para o pino **VERMELHO**. Com isso, o LED azul ficará aceso e o LED vermelho apagado e assim permanecerão.

Na linha 19, temos a função de retardo ***delay***, parando a programação por 500 milissegundos. Após esse tempo a programação passará para a linha seguinte.

A partir da linha 21, temos a mesma programação explicada à dois parágrafos atrás, onde agora temos a função ***digitalWrite*** fazendo o inverso do que foi apresentado, ou seja, apagando o LED azul e acendendo o vermelho. A função ***delay*** na linha 23, manterá os LEDs nesses estados por 500 milissegundos, após isso, a programação retorna para a linha 17 executando tudo novamente até que o Arduino seja desligado.

FUNCIONAMENTO

Assim que o Arduino for ligado os LEDs começarão a piscar em modo alternado, ou seja, enquanto um estiver aceso o outro estará apagado. Como podemos perceber analisando o programa, o LED azul sempre começará aceso e o vermelho apagado, ficarão assim por 500 milissegundos (meio segundo) e depois trocarão de situação, onde agora o LED vermelho acenderá e o azul apagará e ficam assim a cada meio segundo, piscando alternadamente, o que nos fará lembrar de um giroflex de carro policial.

SEQUENCIAL DE LEDS

Objetivo

A ideia seguinte mostrará uma maneira prática e rápida de fazer a configuração de diversas portas digitais com poucas linhas de código, além de nos possibilitar o estudo de acionamentos de cargas em modo sequencial. Aqui serão apresentados dois projetos que usarão o mesmo circuito eletrônico, apenas o firmware (programa) será substituído.

Montagem do circuito

O circuito abaixo servirá para ser usado com os dois programas que iremos trabalhar nesse projeto.

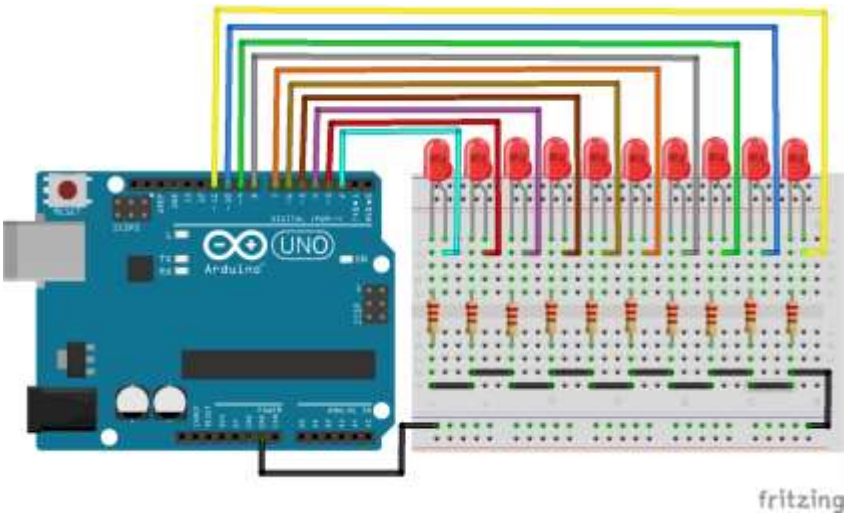


Figura 4 - *Circuito Sequencial de LEDs.*

Lista de materiais

Arduino UNO
10 LED vermelho
10 Resistor 220Ω (vermelho, vermelho, marrom)
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

Sketch 1

Na Arduino IDE, escreva o código abaixo para realizarmos nosso primeiro experimento sobre o tema proposto.

```
1  /************************************
2   * ------ Sequencial de LEDs ------ *
3   ************************************/
4
5  void setup() {
6
7
8    for(int i = 2; i < 12; i++)
9    {
10     pinMode(i, OUTPUT);
11   }
12
13 }
14
15 void loop() {
16
17
18   for(int i = 2; i < 12; i++)
19   {
20     digitalWrite(i, HIGH);
21     delay(300);
22   }
23
24
25   for(int i = 2; i < 12; i++)
26   {
27     digitalWrite(i, LOW);
28     delay(300);
29   }
30
31 }
```

EXPLICANDO O SKETCH 1

A novidade nesse código inicia-se dentro da função *setup()*. Note, na linha 8, a presença de um laço de repetição *for* e entre suas chaves, na linha 10, a função *pinMode* determinando o pino *i* como saída digital. Mas afinal de contas, quem é esse pino *i*?

Sabemos que o laço *for* vai incrementando o valor de *i* a partir de um valor inicial. Na linha 8, podemos ver que a variável *i* iniciou com o valor 2, esse valor é o número correspondente ao pino que controlará o LD1 de acordo com o esquema da figura 1, sendo assim, como *i* inicialmente representa o número 2, logo, ***pinMode*** estará fazendo com que esse pino se torne uma saída digital por meio do comando ***OUTPUT*.** Como a variável *i* vai incrementando seu valor enquanto for menor que 12, a função ***pinMode***, automaticamente, vai tornando os demais pinos da sequência depois do **2 *(3, 4, 5, 6, 7, 8, 9, 10 e 11)*** como saídas digitais. Quando *i* chegar a 12, a função *for* deixa de contar e todos os pinos citados estarão configurados como saídas digitais e o programa segue para a função *loop()*.

Observe, a partir da linha 18, que foi utilizado o mesmo recurso explicado no parágrafo anterior, com a diferença de que estamos usando a função ***digitalWrite*** entre as chaves do *for* ao invés de ***pinMode***. Note que a cada incremento da variável *i*, um LED irá acender em sequência, visto que, o comando usado por ***digitalWrite*** é o ***HIGH*.** Ao final da contagem do ***for***, todos os LEDs estarão acesos e o programa segue para a próxima função que está presente na linha 25.

A função na linha 25, segue com a mesma explicação apresentada para a função da linha 18, vista no parágrafo acima, tendo como única diferença o comando ***LOW***, enviado aos LEDs por meio da função ***digitalWrite***. Com isso, o efeito será contrário ao anterior, ou seja, os LEDs irão sendo apagados em sequência. Ao final da contagem do laço *for*, todos os LEDs estarão apagados e o programa retornará para a linha 18 fazendo tudo novamente.

FUNCIONAMENTO COM SKETCH 1

No momento em que o Arduino for ligado, os LEDs começaram a acender em sequência, no sentido de LD1 para LD10, a cada 300 milissegundos. Assim que todos os LEDs estiverem acesos, após 300 milissegundos, começaram a ser apagados um a um na mesma sequência com o mesmo intervalo de tempo. Para alterar a velocidade com a qual acendem e apagam, basta substituir o valor **300**, nas linha 21 e 28 pelo valor desejado, lembrando apenas de que quanto maior o valor, mais lento ficará os efeitos. Após fazer a alteração, envie o código novamente para o Arduino e veja como ficou.

Sketch 2

Como bônus, temos aqui mais um código para ser usado no mesmo circuito montado para o projeto do sketch 1. Escreva o código mostrado abaixo e envie para o Arduino.

```
1  /*************************************
2   * ------ Sequencial Vai-e-Vem ------ *
3   *************************************/
4
5  void setup() {
6
7    for(int i = 2; i < 12; i++)
8    {
9      pinMode(i, OUTPUT);
10   }
11 }
12
13 void loop() {
14
15   for(int i = 2; i < 12; i++)
16   {
17     digitalWrite(i, HIGH);
18
19     if(i > 2) digitalWrite(i - 1, LOW);
20
21     delay(300);
22   }
23
24   for(int i = 11; i > 1; i--)
25   {
26     digitalWrite(i, HIGH);
27
28     if(i < 11) digitalWrite(i + 1, LOW);
29
30     delay(300);
31   }
32 }
```

EXPLICANDO O SKETCH 2

O código proposto é bastante semelhante ao do sketch anterior, portanto, não iremos nos ater na explicação da função contida em *setup()*, sendo que já foi explicado anteriormente no último projeto. Até mesmo na função *loop(),* podemos perceber, na linha 15, a função *for* sendo usada da mesma forma e com o mesmo propósito de fazer os LEDs acenderem em sequência com intervalos de 300 milissegundos. No entanto, na linha 19, encontramos algo diferente. Trata-se da função condicional ***if***.

A função ***if,*** que traduzindo para o português quer dizer “se”, é uma função condicional, ou seja, ela somente executará uma dada tarefa se e somente se as condições definidas entre parênteses forem atendidas, ou tecnicamente falando, se suas premissas forem verdadeiras. Exemplo:

Observe as duas situações seguintes, em “*A”* a variável *Coisa* recebe o valor 2 e em “*B*” a mesma variável está recebendo 3.

A)

```
int Coisa = 2;

if(Coisa < 3){

    Acender luz;

}
```

Ou podemos escrever assim:

```
If(Coisa < 3) Acender luz;
```

B)

```
int Coisa = 3;

if(Coisa < 3){

    Acender luz;

}
```

Ou também podemos escrever assim:

```
If(Coisa < 3) Acender luz;
```

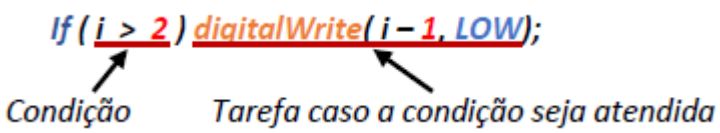
Em "*A*", temos a função *if* comparando se o valor da variável *Coisa* é *menor do que* 3. Caso isso seja verdadeiro, e podemos perceber que é, a tarefa ***Acender luz***, será realizada. Note que existem duas maneiras de escrever a função *if*.

Em "*B*", a variável *Coisa* está com o valor 3 e a função *if* está comparando se esse valor é *menor que* 3. Obviamente, notamos que a sentença é falsa, pois o número três nunca será menor que ele mesmo, portanto, a tarefa ***Acender luz*****,** não será executada.

A função *if* poderá ser usada com os seguintes operadores relacionais, o que dependerá de sua aplicação:

Operador	Comparação
<	Menor que
>	Maior que
==	Igual
!=	Diferente
<=	Menor ou igual
>=	Maior ou igual

Com essas explicações, vamos entender o que temos na linha 19:



Tarefa caso a condição seja

Condição

Lendo esta linha de código no bom português teremos o seguinte:

"Se o valor de i for maior que dois, então apague o LED que está no pino i – 1"

Supondo que *i* tenha seu valor maior que **2**, digamos 3, isso tornará a condição verdadeira, logo, a função ***digitalWrite*** enviará ***LOW*** para o pino correspondente a *i* – ***1.*** Como *i* vale 3, então essa subtração fica sendo **3 – 1 = 2**, ou seja, o LED que será apagado é o LD1, justamente o que está conectado ao pino 2, conforme podemos observar no esquema da figura 1.

Dessa forma, a cada contagem do laço *for*, um LED irá acender em sequência e seu antecessor será apagado pela função *if,* e com isso, teremos sempre apenas um LED diferente aceso a cada 300 milissegundos.

Após o término da contagem do laço *for*, o programa passa para a linha 24, onde agora podemos notar que esse outro laço não irá incrementar o valor de *i*, mas sim irá fazer o contrário, decrementará a cada contagem. Observe que dessa vez *i* foi inicializado com o valor **11**, correspondente ao pino do LED LD10, sendo este o primeiro LED dessa função a ser aceso. Logo em seguida, com o primeiro passo do laço *for*, a variável *i* que estava valendo **11**, agora valerá **10**, correspondente ao pino de LD9 fazendo este ficar aceso, ao passo que o LD10 será apagado pela função *if* da linha 28. Veja que nessa função agora, como estamos decrementando valores, a função ***digitalWrite*** estará realizando uma soma (*i + 1*) para enviar ***LOW*** ao LED de maior valor. Com o fim da contagem desse laço, teremos somente LD1 aceso e os demais apagados e o programa retornará para a linha 15, fazendo tudo novamente e assim os LEDs ficarão indo e voltando.

FUNCIONAMENTO COM SKETCH 2

Inicialmente, ao ligar o Arduino, verá apenas um LED acendendo de cada vez partindo de LD1 até LD10. Ao chegar em LD10, os LEDs voltarão até LD1, da mesma forma como foram, um a um em sequência e com intervalos de 300 milissegundos entre um LED e outro. Altere os valores das funções *delay*, nas linhas 21 e 30, envie novamente para o Arduino e veja o que acontece com a velocidade dos LEDs.

CONTROLE DE LED POR BOTÃO LIGA/DESLIGA

Objetivo

Nesse projeto será apresentada a maneira de configurar uma porta de entrada digital do Arduino para fazer a leitura de botões e contaremos com dois sketches de programação. O princípio de programação e funcionamento que serão apresentados servirão de base para a compreensão a respeito do uso de diferentes tipos de sensores digitais. Será visto também a importância da utilização de resistores nas posições de pull-up e de pull-down.

Montagem do circuito

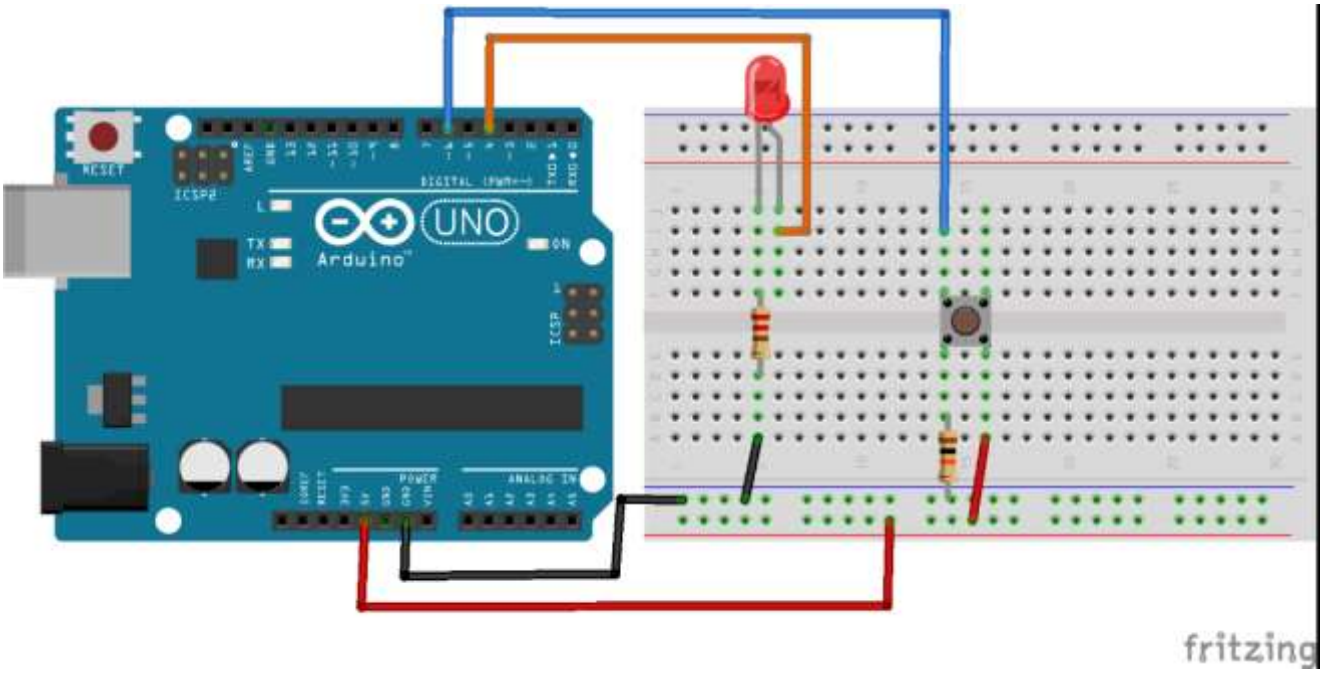


Figura 5 - *Circuito controlado por botão.*

Lista de materiais

Arduino UNO
1 LED vermelho
1 Resistor 220Ω (vermelho, vermelho, marrom) para R1
1 Resistor 10k (marrom, preto, laranja) para R2
1 chave tátil (Botão tipo push button)
Protoboard
Fios ou cabinhos jumpers
Cabo USB

PROGRAMAÇÃO

Sketch 1

```
1  /*****************************************
2   * ------ Controle de LED por botão ------ *
3   *****************************************/
4
5  #define LED 4
6  #define SINAL 6
7
8  bool botao = false;
9
10 void setup() {
11
12   pinMode(LED, OUTPUT);
13   pinMode(SINAL, INPUT);
14
15   digitalWrite(LED, LOW);
16
17 }
18
19 void loop() {
20
21     botao = digitalRead(SINAL);
22     delay(50);
23
24     if(botao == true)
25     {
26       digitalWrite(LED, HIGH);
27     }
28     else
29     {
30       digitalWrite(LED, LOW);
31     }
32
33
34 }
```

EXPLICANDO O SKETCH 1

A programação começa com a definição de nomes para os pinos 4 e 6, conforme podemos observar nas linhas 5 e 6, que já foi explicado sobre os motivos. Na linha 8 temos a declaração de uma variável chamada “botao”, sem acento mesmo, pois, nomes de variáveis não podem conter acentuações ou caracteres especiais, com exceção do underline “_” que pode ser usado para compor o nome da variável. Note que essa variável é do tipo ***bool*****,** esse tipo de variável ocupa apenas um byte em memória podendo armazenar apenas o valor ***zero*** ou o valor ***um***, somente um deles, onde ***zero*** significa falso e *um* significa verdadeiro, sendo assim, podemos substituir o **zero** pela palavra ***false*** (falso em inglês) e o número ***um*** por *true* (verdadeiro em inglês), ambas em letras minúsculas. É o tipo de variável adequado para receber dados de botões, uma vez que eles só enviam para o Arduino *zero* (***LOW***) ou *um* (***HIGH***).

Dentro da função *setup()*, nos deparamos com a função *pinMode* duas vezes. Na linha 12, tal função está configurando o pino 4, que foi nomeado como *LED* para ser uma saída digital através do comando *OUTPUT* e na linha 13, a mesma função está fazendo o pino 6, que recebeu o nome de *SINAL*, ser uma entrada digital, isso por meio do comando *INPUT*. Ainda na função *setup()*, na linha 15, temos a função

digitalWrite enviando nível lógico baixo para o *LED* através do comando *LOW*, isso irá garantir que LD1 sempre inicie apagado no circuito quando o Arduino for ligado.

Na função *loop()* podemos ver, na linha 21, como é realizada a leitura de um sinal vindo por uma entrada digital. Observe que a variável botao está recebendo a função *digitalRead*. Essa função captura o que está na entrada digital passando como parâmetro e entregando o que capturou para a variável botao. Note que como parâmetro dessa função temos o *SINAL*, correspondente ao pino 6 que foi configurado como uma entrada digital, portanto, tudo o que entra por esse pino, sendo nível baixo ou nível alto, será entregue a variável por meio da função de leitura *digitalRead*.

Para garantir a estabilidade do sinal de entrada e minimizar os efeitos do fenômeno conhecido como ***boucing***, o qual ocorre quando um dispositivo mecânico como botões são acionados, temos na linha 22 a função *delay* com um tempo de 50 milissegundos, tempo esse suficiente para reduzir os efeitos do fenômeno descrito e garantir uma determinação melhor do sinal de entrada.

A partir da linha 24 temos a programação que faz a comparação do sinal de entrada com um valor que queremos. Veja que nessa linha temos a função *if*, a qual está comparando se o valor recebido pela variável botao é igual ao valor *true*. Sobre a função *if* e seus operadores foram apresentadas as explicações no projeto anterior. Caso o valor recebido pela variável seja mesmo igual a *true* (nível lógico alto), então a função *digitalWrite*, da linha 26, enviará *HIGH* para acender o LD1. Mas, se a variável receber o valor *false* (nível lógico baixo) em sua entrada? Bom... é para isso que existe a função dependente usada na linha 28, a função *else*. Essa é uma função que depende da função *if* para existir, ou seja, ela não funciona sem antes existir um *if*. Com ela temos o "**senão**" da questão e assim se a variável botao receber *true*, então o LD1 acende, **senão** o LD1 será apagado ou permanecerá apagado. Perceba que se a variável receber *false*, a função *digitalWrite* da linha 30 enviará *LOW* para LD1, fazendo-o permanecer apagado ou apagando se estiver aceso isso graças ao *else*.

Após a comparação da variável, conforme descrito, o programa retorna para a linha 21 para fazer tudo novamente.

Da linha 24 até a linha 30 podemos ler livremente da seguinte forma: "***Se*** botao for ***verdadeiro***, *acender LD1*, ***senão***, *apagar LD1*".

Funcionamento do sketch 1

No momento em que o Arduino for ligado, o LED deverá permanecer apagado, mas no instante em que o botão S1 for pressionado, LD1 deverá acender e permanecer aceso somente enquanto o botão

estiver sendo pressionado, mas quando S1 for liberado, LD1 deverá apagar imediatamente. Para entender melhor como um botão envia nível lógico baixo ou alto, observe a ilustração abaixo:

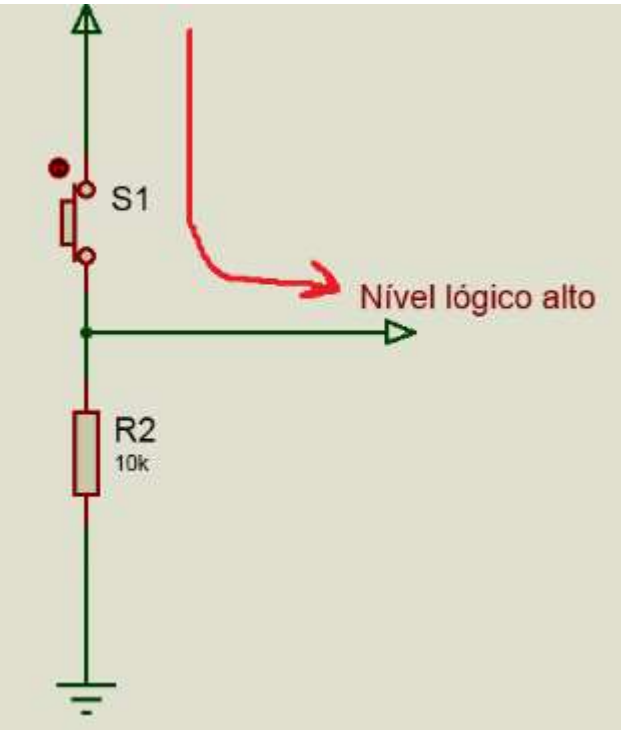


Figura 6 - *Botão pressionado. Nível lógico alto passando pelo botão para o pino 6.*

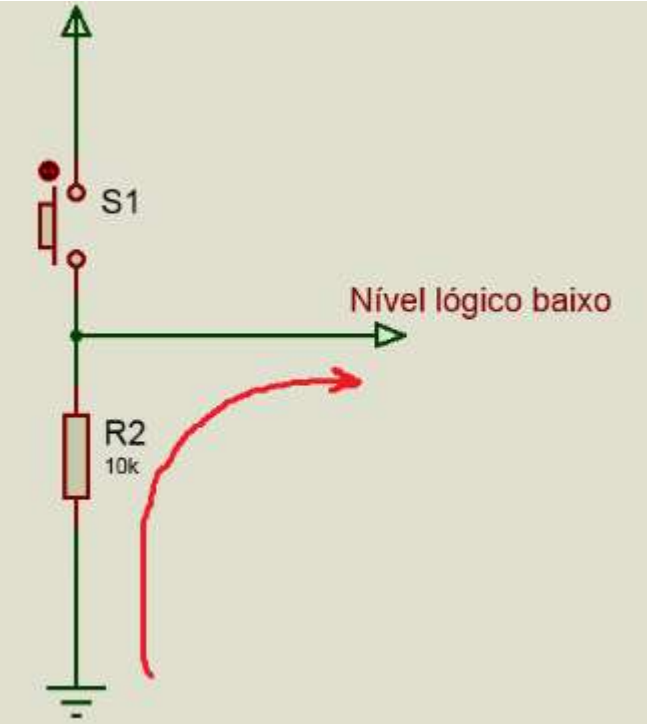


Figura 7 - Botão não pressionado. Nível lógico baixo do GND indo para o pino 6.

Por meio da figura 2 pode-se perceber que o botão não está fechado e o resistor R2 está garantindo que o pino 6 receba ***false***, proveniente do GND do Arduino. Na figura 3, o botão está fechado, e como ele não oferece nenhuma resistência à passagem da corrente elétrica, o pino 6 receberá ***true***, proveniente dos 5 Volts do Arduino. O resistor R2, nessas condições, é chamado de pull-down, por garantir nível lógico baixo enquanto a chave S1 estiver aberta. Caso a posição do resistor R2 fosse conforme a figura 4, ele seria chamado de pull-up, pois estaria garantindo nível lógico alto ao pino configurado como entrada digital enquanto o botão estivesse aberto.

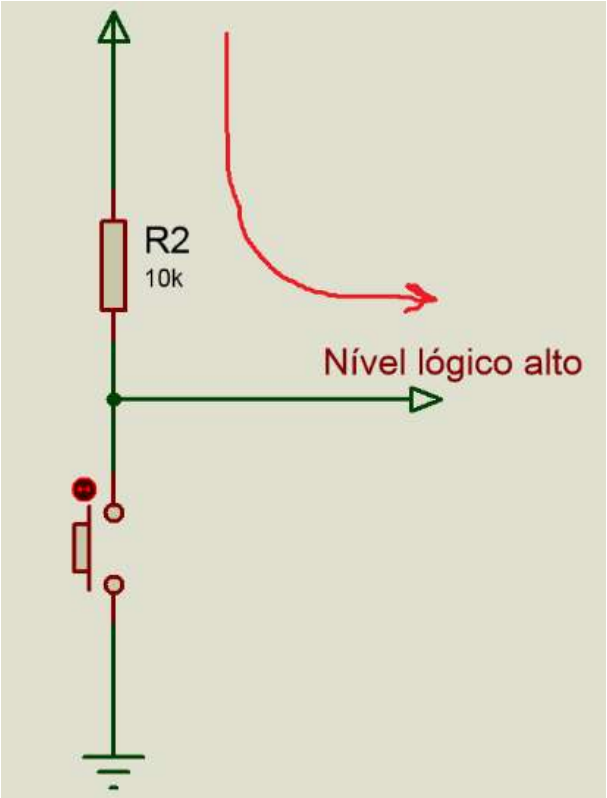


Figura 8 - *Resistor de pull-up, garantindo HIGH para a entrada digital caso o botão esteja aberto.*

Sketch 2

```
1  /****************************************
2   * ------ Liga/Desliga por botão ------ *
3   ****************************************/
4
5  #define LED 4
6  #define SINAL 6
7
8  bool botao = false;
9  bool estado = false;
10
11 void setup() {
12
13   pinMode(LED, OUTPUT);
14   pinMode(SINAL, INPUT);
15
16   digitalWrite(LED, LOW);
17 }
18
19 void loop() {
20
21     botao = digitalRead(SINAL);
22     delay(50);
23
24     if(botao == true)
25     {
26       estado = !estado;
27
28       digitalWrite(LED, estado);
29
30       while(botao) botao = digitalRead(SINAL);
31     }
32
33 }
```

EXPLICANDO O SKETCH 2

O programa apresentado é bastante semelhante ao do projeto anterior, mas note uma nova declaração de variável também do tipo *bool* na linha 9, chamada de "estado" e inicializada com o valor *false*. Essa variável irá armazenar o estado atual de LD1, se ligado ou desligado. As funções dentro de *setup()* já foram apresentadas e explicadas, bem como os conteúdos das linhas 21 e 22. Portanto, vamos nos ater apenas às explicações da função *if* para esse projeto.

Na linha 24 temos a função *if* verificando se a variável botao é igual a *true* (nível lógico alto), mas, observe que dessa vez, caso a verificação seja verdadeira, o LED não receberá simplesmente o *HIGH*, mas sim teremos algo curioso que ocorrerá na linha 26. Vamos entender:

```
estado = !estado;
```

Essa linha nos diz que a variável estado irá receber ela mesma, porém, com valor invertido, por isso o sinal de exclamação após o sinal de igualdade. Como essa variável foi inicializada com o valor *false*, quando o programa passar por essa linha de código, a variável estado irá trocar seu valor para *true*, e quando o programa passar novamente por essa linha, o valor será trocado para *false* e sempre ficará

sendo invertido toda vez que essa linha entrar em foco na programação, isso graças ao sinal de negação representado pela exclamação. Assim, a variável estado estará alternando seu valor entre ***false*** e ***true***, ou podemos também dizer que estará alternando entre ***LOW*** e ***HIGH***.

Toda vez que o botão for pressionado, a função ***digitalWrite***, na linha 28, enviará o valor da variável estado para o pino *LED*, ou seja, se a variável estado estiver portando o valor ***false***, que é o mesmo que **LOW**, então LD1 irá ficar apagado, mas se estiver portando ***true***, que é o mesmo que ***HIGH***, LD1 irá acender. A cada toque no botão teremos um estado diferente do anterior e assim um toque acenderá LD1 e outro toque apagará.

Para evitar que LD1 fique piscando enquanto o botão estiver sendo pressionado, temos a função na linha 30 que resolverá esse problema. A função ***while*** (que em português quer dizer “enquanto”) prenderá o programa na linha 30 e ficará apenas lendo o sinal de entrada e guardando o valor capturado na variável botao da mesma forma como na linha 21, sem deixar que nada mais aconteça e isso ocorrerá enquanto o botão estiver sendo pressionado, ao liberar, o programa volta para a linha 21.

Funcionamento do sketch 2

Ligando o Arduino, LD1 estará apagado, mas quando for dado um toque no botão, LD1 irá acender e assim permanecerá. Com outro toque no botão LD1, irá apagar e permanecerá nesse estado até que o botão seja pressionado novamente. Caso LD1 esteja ligado e o botão seja pressionado e assim se mantenha, o LD1 irá apagar e permanecerá apagado, só voltando a acender quando tirar o dedo do botão e pressionar novamente, o mesmo ocorrerá em situação inversa, ou seja, um toque liga e outro toque desliga.

SEQUENCIAL DE LEDS CONTROLADO POR BOTÃO

Objetivo

Nesse projeto iremos praticar um pouco mais sobre os conceitos apresentados em relação ao uso de botões e o acionamento de cargas em sequência, no entanto, veremos como realizar a escolha de opções e mudar o efeito sequencial dos LEDs. Isso será extremamente útil para compreender como são criados os menus de opções em equipamentos eletrônicos microcontrolados. Outro assunto importante que iremos aprender é o desenvolvimento de funções para otimização dos códigos.

Montagem do circuito

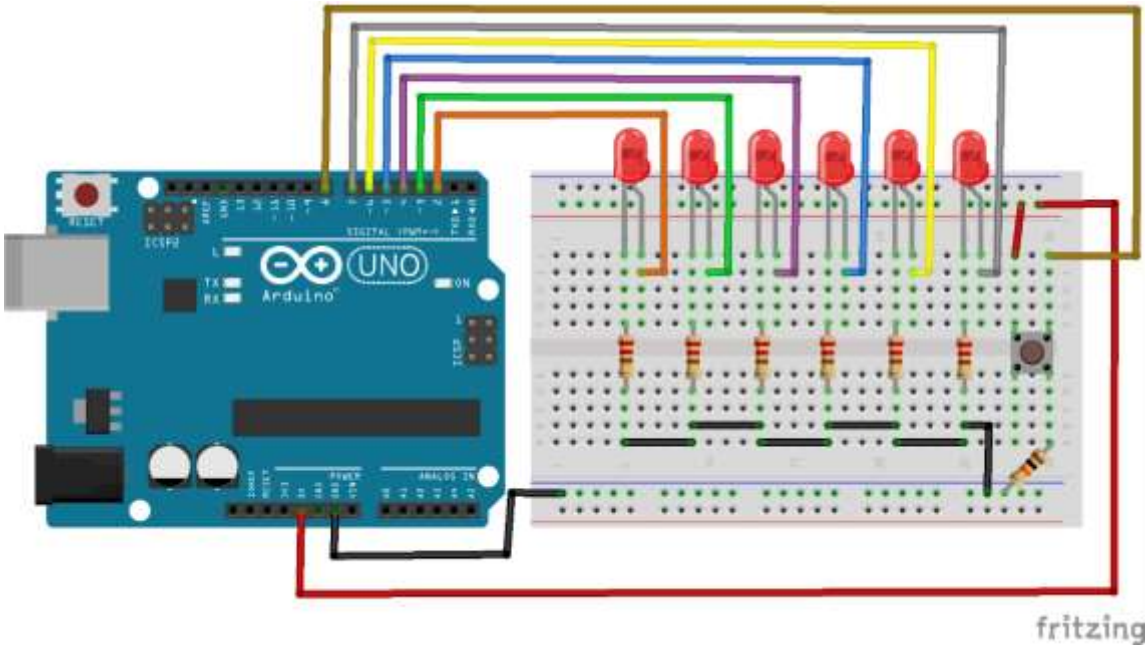


Figura 9 - *Montagem do sequencial de LEDs controlado por botão.*

Lista de materiais

Arduino UNO
6 LED vermelho
6 Resistor 220Ω (vermelho, vermelho, marrom)
1 Resistor 10kΩ (marrom, preto, laranja)
1 Botão tátil do tipo Push-button
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

A programação para esse projeto será apresentada em quatro partes por ser maior que as já estudadas anteriormente. Portanto, teremos a primeira parte com as definições, declarações de funções e de variáveis e a função setup(); logo em seguida, na segunda parte, teremos a função loop(); a terceira parte com duas das quatro funções que iremos desenvolver fora das funções setup() e loop() e que acenderão os LEDs em sequências distintas e, para finalizar, na quarta parte teremos a função que desligará os LEDs e a função de leitura do botão.

Código completo

Na Arduino IDE, digite o código apresentado abaixo começando da linha 5 até a 104.

```
/*************************************************
 * ------ Sequencial Controlado por botão ------ *
 *************************************************/

#define SINAL 8

void Acende_e_apaga();
void Desligar();
void Ler_botao();
void Vai_e_vem();

bool botao = false;
int option = 0;

void setup() {

  for(int i = 2; i < 8; i++)
  {
    pinMode(i, OUTPUT);
  }

  pinMode(SINAL, INPUT);
}

void loop() {

  Ler_botao();

  switch(option)
  {
    case 0:
      Desligar();
    break;

    case 1:
      Vai_e_vem();
    break;

    case 2:
      Acende_e_apaga();
    break;
  }

}
```

```
void Vai_e_vem()
{
  for(int i = 2; i < 8; i++)
  {
    digitalWrite(i, HIGH);

    if(i > 2) digitalWrite(i - 1, LOW);
    Ler_botao();
    delay(300);
  }

  for(int i = 8; i > 1; i--)
  {
    digitalWrite(i, HIGH);

    if(i < 8) digitalWrite(i + 1, LOW);
    Ler_botao();
    delay(300);
  }
}
```

```
67  void Acende_e_apaga()
68  {
69    for(int i = 2; i < 8; i++)
70    {
71      digitalWrite(i, HIGH);
72      Ler_botao();
73      delay(300);
74    }
75
76    for(int i = 2; i < 8; i++)
77    {
78      digitalWrite(i, LOW);
79      Ler_botao();
80      delay(300);
81    }
82  }
```

```
84  void Desligar()
85  {
86    for(int i = 2; i < 8; i++)
87    {
88      digitalWrite(i, LOW);
89      Ler_botao();
90    }
91  }
92
93  void Ler_botao()
94  {
95    botao = digitalRead(SINAL);
96    delay(50);
97
98    if(botao == true)
99    {
100     option++;
101     if(option > 2) option = 0;
102     while(botao) botao = digitalRead(SINAL);
103   }
104 }
```

EXPLICANDO O SKETCH

Primeira parte

A novidade que encontramos no início desse programa, vendo as linhas de código abaixo, está nas linhas de 7 a 10 e que corresponde na declaração de funções, também chamado de protótipo de funções. Antes de criarmos uma função devemos definir o tipo e um nome para ela, sendo que esse nome segue as mesmas regras usadas para nomes de variáveis. No momento veremos apenas funções do tipo ***void***, o que significa que serão funções que não retornaram com nenhum valor, mas que irá

executar uma tarefa definida pelo desenvolvedor. Nas linhas seguintes é possível notar a presença de elementos já estudado, bem como o conteúdo da função *setup()*, portanto, não iremos explanar maiores detalhes.

```
1  /*************************************************
2   * ------ Sequencial Controlado por botão ------ *
3   *************************************************/
4
5  #define SINAL 8
6
7  void Acende_e_apaga();
8  void Desligar();
9  void Ler_botao();
10 void Vai_e_vem();
11
12 bool botao = false;
13 int option = 0;
14
15 void setup() {
16
17   for(int i = 2; i < 8; i++)
18   {
19     pinMode(i, OUTPUT);
20   }
21
22   pinMode(SINAL, INPUT);
23 }
```

SEGUNDA PARTE

A função *loop()* vem trazendo as principais novidades desse projeto. Na linha 27 encontramos uma das funções definidas na declaração. O ato de estar presente na parte de execução do programa é conhecido como chamada de função, sendo assim, na linha 27 estamos chamando a função ***Ler_botao()***. Mas para que essa função realmente sirva para fazer algo devemos desenvolvê-la e isto é feito fora da função *loop()*, lá na parte de baixo. Observe o código apontado pela seta, veja que esse código inicia-se na linha 93 e corresponde ao desenvolvimento da função ***Ler_botao()***.

No desenvolvimento da função ***Ler_botao()***, logo na linha 95, nos deparamos com a função que realiza a leitura de botões, algo que já estudamos no projeto *Controle de LED por botão liga e desliga*, onde já tecemos as explicações e práticas. Sobre a função *if* da linha 98, também já realizamos um estudo no mesmo projeto citado. Perceba, analisando os códigos da linha 95 até a linha 102, que toda vez que o botão S1 for pressionado, a variável *botao* receberá o valor *true* e toda vez que isso ocorre, a variável *option*, na linha 100, será incrementada em mais um. Na linha 101 temos uma espécie de limitador de valor e com ele se a variável atingir um valor maior de dois, voltará para zero, sendo assim, os valores

possíveis para a variável option durante cada incremente são 0, 1 e 2, nada mais além desses valores, ou seja, a cada toque em S1 um desses valores é assumido em sequência pela variável e quando chega a dois o ciclo se repete. A função na linha 102 é apenas para evitar que a variável *option* troque de valor por mais de uma vez, segurando o código a cada pressionar de S1.

Com isso, entenda que toda vez que um programa encontra uma função no meio do caminho, ele entra nela e executa tudo o que tem nela. É como se, no caso do nosso exemplo, quando o programa chegar na linha 27 ele saltasse para a linha 95, executando tudo em sequência e depois retorna para a linha 29, onde está a função *switch*.

A partir deste ponto vamos entender como funciona a função *switch*, a qual é mais uma das quatro funções fundamentais presente na maioria das linguagens de programação. Observe na linha 29 a seguinte escrita: *switch(option)*. Isto significa que a função estará olhando para o valor que a variável option irá assumir, dependendo do valor dessa variável e função irá selecionar um, dentre várias opções presente entre suas chaves. Da linha 31 até a 33 temos a primeira opção, ou podemos dizer, temos o primeiro caso. A linha 31 nos diz que se caso o valor da variável seja zero, notamos isso pela expressão *case: 0*, na linha 32 será chamada a função *Desligar()* e na linha 33 temos o comando *break* que segura e impede que o *switch* execute as outras opções de forma indesejada. Sendo assim, fica fácil entender que dependendo do valor da variável *option*, a função irá selecionar a tarefa correspondente ao *case* que contém tal valor.

Para esse projeto, a variável *option* poderá assumir apenas três valores: 0, 1 e 2, por ser estes referentes ao número de tarefas que temos para serem executadas dependendo da opção escolhida e por isso, nossa função *switch* obteve apenas três blocos de *case/break*. No entanto, em outros projetos que o aluno venha a desenvolver, poderá usar a função *switch* com a quantidade de *case/break* que precisar.

EXERCÍCIO

Como forma de exercitar o que aprendeu até o presente momento e tendo em vista que as próximas funções contêm apenas elementos que já foram explicados em projetos anteriores, procure ler cada linha e tente entender como cada função irá se comportar quando for chamada. As funções que deverá entender, são:

Vai_e_vem();
Acende_e_apaga();

Desligar().

A função Ler_botao() já foi explicada linhas atrás.

```
46  void Vai_e_vem()
47  {
48    for(int i = 2; i < 8; i++)
49    {
50      digitalWrite(i, HIGH);
51
52      if(i > 2) digitalWrite(i - 1, LOW);
53      Ler_botao();
54      delay(300);
55    }
56
57    for(int i = 8; i > 1; i--)
58    {
59      digitalWrite(i, HIGH);
60
61      if(i < 8) digitalWrite(i + 1, LOW);
62      Ler_botao();
63      delay(300);
64    }
65  }
66
67  void Acende_e_apaga()
68  {
69    for(int i = 2; i < 8; i++)
70    {
71      digitalWrite(i, HIGH);
72      Ler_botao();
73      delay(300);
74    }
75
76    for(int i = 2; i < 8; i++)
77    {
78      digitalWrite(i, LOW);
79      Ler_botao();
80      delay(300);
81    }
82  }
```

```
84  void Desligar()
85  {
86    for(int i = 2; i < 8; i++)
87    {
88      digitalWrite(i, LOW);
89      Ler_botao();
90    }
91  }
92
93  void Ler_botao()
94  {
95    botao = digitalRead(SINAL);
96    delay(50);
97
98    if(botao == true)
99    {
100     option++;
101     if(option > 2) option = 0;
102     while(botao) botao = digitalRead(SINAL);
103   }
104 }
```

FUNCIONAMENTO

Ao energizar o Arduino os LEDs estarão todos apagados esperando o botão ser pressionado. Quando S1 for pressionado pela primeira vez, os LEDs assumirão o comportamento de um vai-e-vem partindo do LD1 até o LD6 e depois voltando de LD6 para o LD1 e repetindo esse ciclo enquanto o Arduino estiver ligado ou enquanto o botão não for pressionado. Apertando novamente o botão o comportamento dos LEDs deverá mudar e agora eles partirão de LD1 para LD6, mas não fazendo o caminho inverso, no entanto, o ciclo se repetirá sempre ressurgindo de LD1 para LD6.

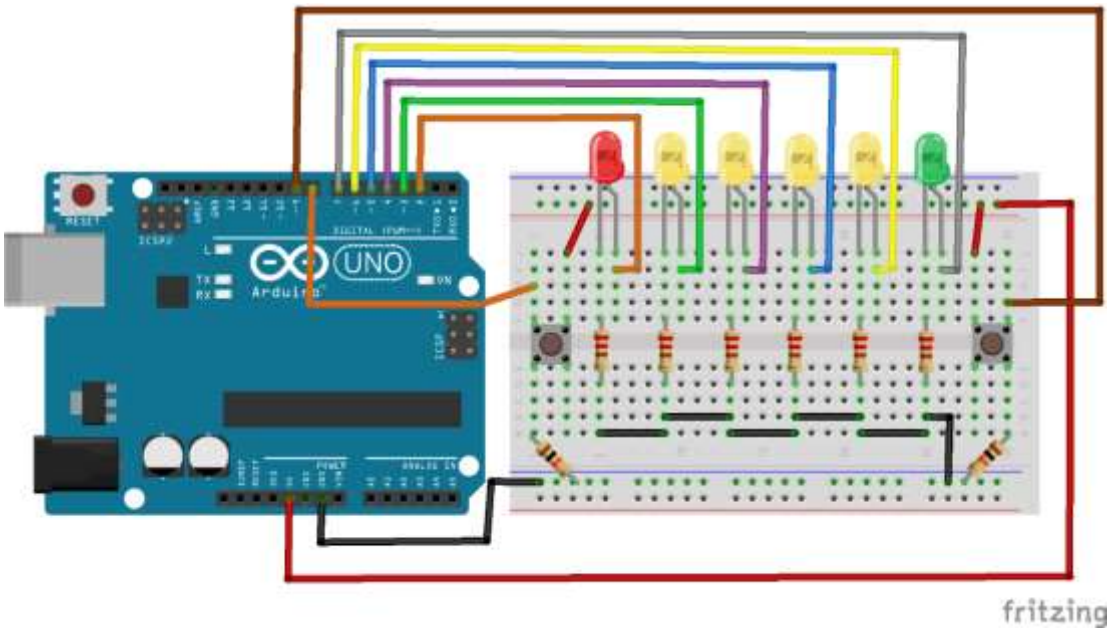
JOGO DE PING-POING

Objetivo

Este projeto tem a finalidade de fazer o aluno praticar grande parte do que já aprendeu até aqui. Consiste em jogo de Ping-Poing com LEDs, onde cada jogador deverá pressionar o botão de sua vez para mudar a direção dos LEDs, caso o botão não seja apertado no momento em que “a bolinha” chegar próximo ao botão, o jogo para e o jogador que deveria ter pressionado o botão perde.

Não teceremos grandes explicações a respeito do funcionamento e do código, uma vez que as funções usadas no desenvolvimento desse jogo já foram todas estudadas.

Montagem do circuito



Lista de materiais

Arduino UNO
1 LED vermelho
1 LED verde
4 LEDs amarelo
6 Resistor 220Ω (vermelho, vermelho, marrom)
2 Resistor 10kΩ (marrom, preto, laranja)
2 Botão tátil do tipo Push-button
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

A programação completa está disponível ao lado.

```
/*********************************
 * ------ JOGO PING-PIONG ------ *
 *********************************/

int bola = 1;
int tempo = 500;

bool jogador_1 = false;
bool jogador_2 = false;

void setup() {

  for(int i = 2; i < 8; i++)
  {
    pinMode(i, OUTPUT);
  }

  pinMode(8, INPUT);
  pinMode(9, INPUT);

}
```

```
void loop() {

  if(bola == 1)
  {
    bola = 0;
    for(int i = 2; i < 8; i++)
    {
      digitalWrite(i, HIGH);

      jogador_2 = digitalRead(9);
      jogador_1 = digitalRead(8);

      if(i > 2)
      {
        digitalWrite(i-1, LOW);
      }

      if(jogador_2 == true && jogador_1 == false && i == 6)
      {
        bola = 2;
      }
      else if(jogador_1 == true && jogador_2 == true)
      {
        bola = 0;
        digitalWrite(2, HIGH);
        digitalWrite(7, LOW);
      }
      delay(tempo);
    }
  }
```

```
53
54    if(bola == 2)
55    {
56      bola = 0;
57      for(int i = 7; i > 1; i--)
58      {
59        digitalWrite(i, HIGH);
60
61        jogador_1 = digitalRead(8);
62        jogador_2 = digitalRead(9);
63
64        if(i < 7)
65        {
66          digitalWrite(i+1, LOW);
67        }
68
69        if(jogador_1 == true && jogador_2 == false && i == 2)
70        {
71          bola = 1;
72        }
73        else if(jogador_2 == true && jogador_1 == true)
74        {
75          bola = 0;
76          digitalWrite(7, HIGH);
77          digitalWrite(2, LOW);
78        }
79        delay(tempo);
80      }
81    }
82
83    tempo -= 50;
84    if(tempo <= 50) tempo = 50;
85
86  }
```

EXPLICANDO O SKETCH

Como toda a parte de configuração desse código já é de conhecimento do aluno, vamos explicar apenas a parte referente à função *loop()*. Observe atentamente que temos duas funções *if* comparando o valor da variável *bola* nas linhas 25 e 54. Cada uma traz entre suas chaves a programação de um sequencial de LEDs visto em projeto anterior, onde sempre haverá apenas um LED aceso ao passo de cada sequência. O diferencial aqui está entre as linhas 40 e 50 e também entre as 69 e 83. Entre essas linhas temos a programação que fica esperando os jogadores apertarem o botão no momento certo. Observe que se os dois jogadores permanecerem com os botões pressionados quando não for a vez de um só jogar, o jogador que não deveria pressionar o botão perderá o jogo, fazendo “a bolinha” voltar e o jogo irá parar.

Na linha 83 temos o decremento da variável *tempo*, o que significa que a cada ciclo de vai-e-vem “da bolinha” o tempo vai ficando menor e o jogo vai aumentando de velocidade. A linha 84 limita o menor tempo em 50 milissegundos.

FUNCIONAMENTO

Quando o Arduino for energizado, os LEDs seguirão em sequência acendendo apenas um a cada passo. A suposta bolinha partirá de LD1 até LD6. Quando LD6 acender, o jogador que estiver com o botão S2

deverá estar pressionando-o para fazer a “bolinha” voltar. Quando a bolinha chegar em LD1, o botão S1 deverá estar pressionado por seu jogador.

Se durante o momento em que a “bolinha” chegar em uma das extremidades e os dois jogadores estiverem pressionando suas teclas, o jogador que não deveria estar apertando o botão perderá a bolinha voltará para o seu lado e o jogo encerra.

Toda vez que um jogador não pressionar o botão a tempo, perderá e com isso o jogo se encerra garantindo a vitória para o outro jogador. Para iniciar outro ciclo terá que pressionar o botão de RESET da placa Arduino.

SENSOR DE MOVIMENTO

Objetivo

O objetivo desse projeto é fazer o aluno perceber que praticamente o mesmo código usado na programação de leituras de botões, podem ser usados para realizar a leitura de sensores digitais, como é o caso do sensor piroelétrico que será apresentado logo mais.

Montagem do circuito

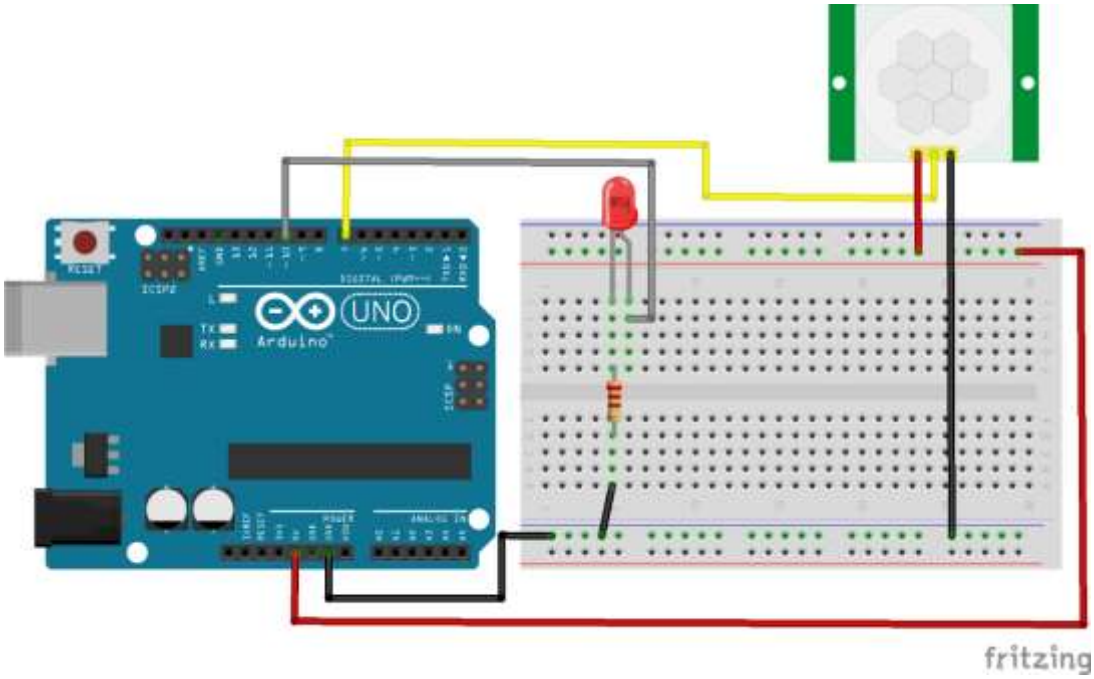


Figura 10 - Circuito para detecção de movimento humano.

Dependendo do sensor adquirido o terminal de 5V e o terminal de GND poderão vim invertidos em relação ao mostrado na figura 1. A figura 2 mostra os terminais. Uma forma de ver qual é a correta

posição dos terminais do sensor que você possui é só remover a tampa do sensor e ver os nomes dos terminais escrito na placa. A figura 3 ilustra isso.

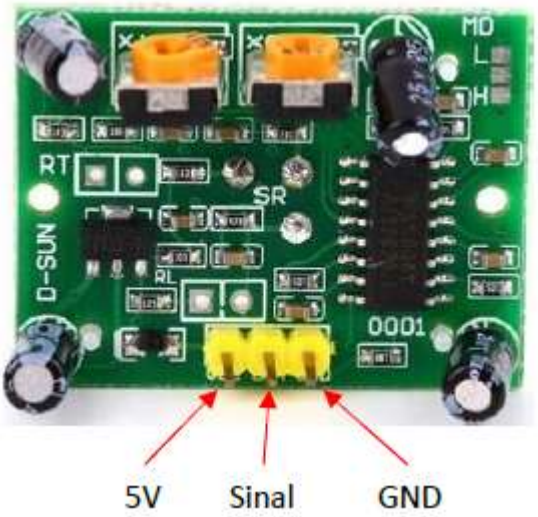


Figura 11 - *Terminais do sensor PIR.*

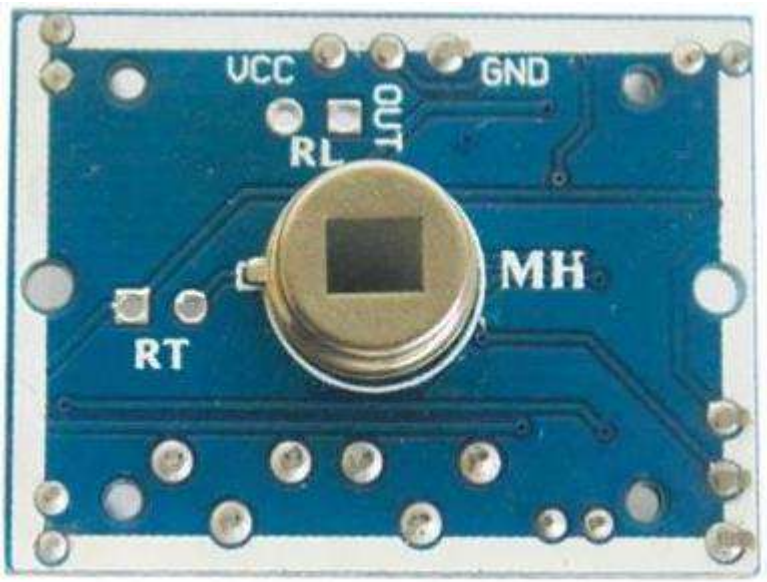


Figura 12 - *Sensor sem a tampa mostrando o nome dos terminais.*

Lista de materiais

Arduino UNO
1 LED vermelho
1 Resistor 220Ω (vermelho, vermelho, marrom)
1 Sensor piroelétrico - PIR
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

A programação para o circuito da figura 1 é muito simples e segue abaixo.

```
/*****************************************
 * ------ Sensor de Movimento PIR ------ *
 *****************************************/

#define LED 10
#define SINAL 7

bool sensor = false;

void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(SINAL, INPUT);
}

void loop()
{
  sensor = digitalRead(SINAL);

  if (sensor == LOW)
  {
    digitalWrite(LED, LOW);
  }
  else
  {
    digitalWrite(LED, HIGH);
  }
}
```

EXPLICANDO O SKETCH

Como pode ver, a programação apresentada é muito simples e pode ser entendida como um simples acionar de botão. Todas as configurações são de conhecimento do aluno, bem como a parte de leitura do sensor, que se olhar com atenção, perceberá que é o mesmo que fizemos para a leitura de um botão. Isso é possível porque o sensor PIR entrega um sinal digital da mesma forma como um botão que já estudamos.

O acionamento do LED é realizado da mesma forma como fizemos para acender um LED com o pressionar de um botão, apagando quando o botão deixar de ser pressionado. Aqui não temos botão, mas note que quando o sensor enviar *LOW*, o LED ficará apagado indicando, com isso, que não foi detectado movimento no ambiente, mas quando o sensor enviar *HIGH*, o LED acenderá, indicando que houve movimento.

Como esse sensor não é um botão, veja que não precisamos nos preocupar em colocar a função *delay* com um tempo pequeno para eliminar o efeito Bouncing, uma vez que o sensor não possui acomodações mecânicas causadoras desse efeito indesejado.

Funcionamento

Ao ligar o Arduino, o sensor levará cerca de três segundos para iniciar a leitura do ambiente. A maioria dos sensores desse tipo poderá ter um alcance de aproximadamente 7 metros com abertura de ângulo de detecção em torno de 120°. A detecção do movimento sempre ocorre no sentido horizontal, em movimentos verticais a detecção é deficiente.

Com o circuito ligado e o sensor já em funcionamento, toda vez que um indivíduo se mover em frente ao sensor, o LED LD1 irá piscar e quando o movimento para o LED permanecerá apagado.

Após uma detecção, o sensor leva um tempo para fazer uma nova leitura e isso pode ser ajustado em um de seus trimpots presentes na placa. O outro trimpot corresponde ao ajuste de sensibilidade que diz respeito à distância máxima. Para iniciar o ajuste, coloque os trimpots em seus valores mínimos, para isso, gire-os todo para o sentido anti-horário e, aos pouco, vai aumentando até chegar no tempo e na sensibilidade adequados ao seu projeto.

Figura 13 - *Ajustes de tempo e sensibilidade.*

MEDINDO VELOCIDADE COM SENSOR HALL

Objetivo

Com esse projeto firmaremos a compreensão a respeito da programação de circuitos que usam sensores digitais. E o sensor dessa vez é o de efeito Hall, muito usado na detecção de campos magnéticos e usaremos essa característica para implementarmos um medidor de velocidade para objetos que contenham um ou mais ímãs presos em sua parte móvel. O aluno verá também a aplicação de fórmulas matemática em um código de programação, além de começar a tomar conhecimento sobre comunicação serial.

Montagem do circuito.

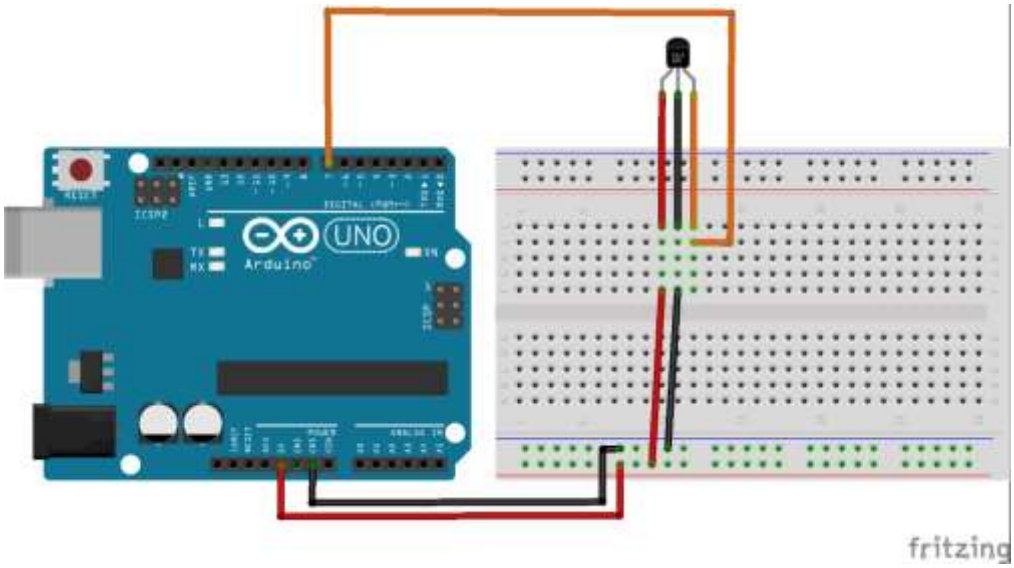


Figura 14 - Circuito do sensor de velocidade.

A figura abaixo mostra o pinout do sensor de efeito Hall:

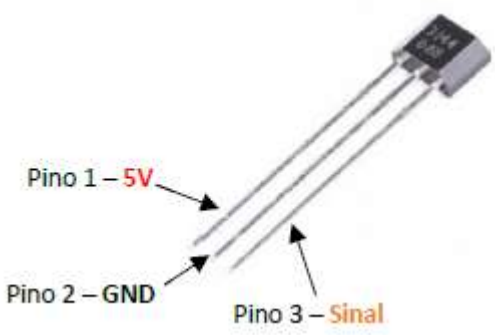


Figura 15 - *Função dos pinos do sensor Hall.*

Observação: O pino de Sinal sempre enviará ao Arduino o nível lógico baixo (false) toda vez que for detectado um campo magnético. Quando não estiver detectando, o pino de sinal permanecerá em nível alto (true).

Lista de materiais

Arduino UNO
1 Sensor Hall 44E ou outro similar
Protoboard
Fios ou cabinhos jumpers
Cabo USB
1 Ímã comum

Programação

O programa completo está disponível logo abaixo, basta escrevê-lo na Arduino IDE e enviar para a placa Arduino e verificar seu funcionamento.

```
/*******************************************************
 * ------ Medida de velocidade com sensor Hall ------ *
 *******************************************************/

#define HALL 7

float circunf = 12;
float distancia;
float tempo;
float tmp;
float veloc;

int ponto = 0;
bool sensor = true;

void setup() {
  Serial.begin(9600);
  pinMode(HALL, INPUT);
}

void loop() {

  sensor = digitalRead(HALL);

  if(sensor == false && ponto == 0)
  {
    tmp = millis();
    ponto = 1;
    while(sensor == false) sensor = digitalRead(HALL);
  }
  else if(sensor == false && ponto == 1)
  {
    tempo = (millis() - tmp)/1000;
    ponto = 2;

    distancia = circunf/100;
    veloc = distancia/tempo;

    Serial.print("Velocidade: ");
    Serial.print(veloc);
    Serial.println("m/s");
    while(sensor == false) sensor = digitalRead(HALL);
  }

  if(sensor == true && ponto == 2) ponto = 0;
}
```

Explicando o sketch

A programação desse sketch traz diversas novidades para o aluno. Logo de início podemos notar a presença de declarações de variáveis com um tipo que ainda não foram usadas em nenhum de nossos projetos, trata-se do tipo *float*. Este tipo de variável serve para armazenar valores que contenham casas decimais, ou seja, os números reais. Ela ocupa muito espaço em memória e para se ter uma ideia, uma variável do tipo *bool* ocupa 1 Byte, podendo aceitar apenas o valor zero ou o valor um, enquanto que uma variável do tipo *float* ocupa 4 Bytes, sendo 24 bits para a mantissa e 8 bits para a parte exponencial. Portanto, é recomendável usar esse tipo de variável somente quando for realmente necessário, como é o caso desse nosso projeto, onde iremos apresentar o valor de velocidade e tal valor é melhor representado com suas casas decimais.

Na função *setup()* encontramos outra novidade, a função ***Serial.begin(xxxx)***. Essa função configura a comunicação serial do Arduino e com isso podemos enviar e receber dados entre um computador e nosso projeto. Essa função inicializa a comunicação serial e para isso só precisamos passar para ela qual a velocidade de comunicação que queremos usar transmitir e receber os dados.

O valor *9600* presente como parâmetro da função *Serial.begin(9600)*, corresponde à velocidade em *bps* (bits por segundo) da comunicação serial, com isso iremos enviar nossas informações para serem mostradas no computador a uma velocidade de *9.600bps*.

Dentro da função *loop()* temos diversos recursos que já foram estudados em outros projetos, logo, iremos nos preocupar apenas com as linhas que contenham novidades. Sugiro que após as explicações inicie uma leitura de todo o código para melhor assimilação de tudo o que já aprendeu até o momento.

Em física, na escola, aprendemos que para calcularmos a velocidade média de um corpo precisaremos da distância percorrida e do tempo que levou para o corpo se deslocar de um ponto inicial até um ponto final. A fórmula para isso é a seguinte:

$$Vm = \frac{S_f - S_i}{t}$$

Onde:

Vm=$Velocidade\ média$;

S_f=$Posição\ final$;

S_i=$Posição\ inicial$;

t=$Tempo\ percorrido\ durante\ o\ trajeto$.

Olhando para a fórmula apresentada, percebemos que iremos precisar de alguma função que marque o tempo decorrido para que possamos chegar ao resultado esperado. Para realizar essa captura de tempo usaremos a função ***millis()*.** Essa função não recebe nenhum parâmetro e quando chamada, passa a contar o tempo de execução do Arduino, ou seja, ela marca o tempo em que o Arduino está em funcionamento. Esse tempo é dado em milissegundos e não é infinito, o que significa que quando atingir o limite de aproximadamente 4,32x109 milissegundos (50 dias) com o Arduino ligado, essa função irá zerar e começará a contar a partir do zero novamente.

Para melhor compreender esse projeto, suponha que iremos medir a velocidade da roda de um veículo. Veja a lustração abaixo para melhor entender:

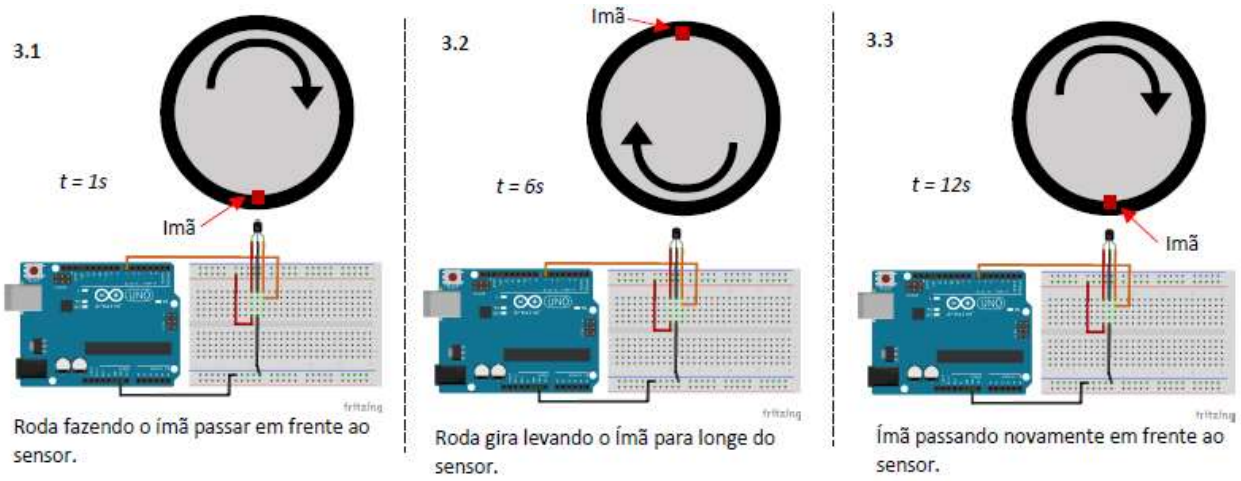


Figura 16 - *Roda movimentando um ímã em frente ao sensor Hall.*

Com isso ficará fácil entender o que temos nas linhas 27 e 33. A linha 27 pertence à parte do código que realizará a leitura do ponto inicial de partida do objeto que estará em movimento, que de acordo com a figura 3, corresponde à imagem 3.1. Essa leitura é dada da seguinte forma: a função *if* da linha 25 fica observando se a variável *sensor* recebeu o valor *false*, o que ocorre quando o ímã passa em frente ao sensor, e também verifica se a variável *ponto* está com o valor zero. Com isso, o comando na linha 27 captura o tempo presente em *millis()* naquele exato momento e armazena na variável *tmp*, esse será nosso tempo inicial. Logo em seguida a variável *ponto* recebe o valor 1, para evitar que na próxima passagem do ímã essas ações voltem a serem executadas. Para evitar contagens erradas com o ímã parado em frente ao sensor, na linha 29 e 42, temos a função *while* que tem a finalidade de paralisar a programação enquanto o ímã estiver parado.

Quando o ímã estiver longe, o pino de sinal do sensor manterá nível lógico alto no pino 7 do Arduino e com isso nada acontece. Essa representação é ilustrada na imagem 3.2 da figura 3.

No momento em que o ímã passar pela segunda vez em frente ao sensor, o código realizará os comandos do *else if* da linha 31. Note que aqui é analisado se a variável *sensor* está com o valor *false*, mas, também verifica se agora a variável *ponto* está com o valor 1. O sensor estando com o ímã em sua frente e como a variável *ponto* havia recebido o valor 1 na linha 28, então isso satisfaz as condições exigidas e os comandos entre as linhas 33 e 42 passam a serem executadas sequencialmente.

Na linha 33, a variável tempo recebe o tempo final do percurso, veja que temos:

$$tempo = (millis() - tmp) / 1000$$

A expressão ***(millis()*** – *tmp)* pega o valor atual de ***millis()*** e subtrai com o valor da variável *tmp* obtido na partida inicial do ímã. Sendo assim, suponha que ***millis()*** esteja nesse momento em 12.000 milissegundos e que a variável *tmp* estivesse armazenado o tempo inicial de 1.000 milissegundos. Com esse subtração, a variável *tempo* receberá o valor de tempo igual a 11.000 milissegundos. Como queremos a unidade de medida de velocidade em metros por segundo (m/s), logo devemos converter esses milissegundos em segundos e é por isso que tal expressão é dividida por 1000, assim o valor final armazenado na variável *tempo* será de 12 segundos. De acordo com a figura 3, isso ocorrerá na imagem 3.3, que é a representação de quando o ímã volta a passar em frente ao sensor.

Com o tempo em segundos já em mãos, precisaremos apenas da distância linear percorrida pelo ímã para calcularmos a velocidade em m/s. O ímã efetuou um movimento circular preso em uma roda, para encontrar a trajetória linear de um ponto que se move em círculo precisamos calcular a circunferência dessa roda. Circunferência é o comprimento dado à reta originária da abertura de um círculo. Para isso, calculamos com a seguinte fórmula:

$$C = \pi \times r$$

Onde:

C=$Circunferência$;

π=$Constante\ de\ aproximadamente$ 3,1415;

r=$raio\ do\ circulo$.

Mas, calma que dessa vez não iremos colocar essa fórmula na programação, ao invés disso, foi dado um valor fictício de 12 cm para a variável *circunf*, lá na linha 7, logo na inicialização. Apresentamos a fórmula para o caso de o aluno aplicar esse projeto em algum outro trabalho.

Como a velocidade será dada em m/s, então teremos que converter esses 12 cm em metros e para isso temos os comandos da linha 36 que divide o valor da variável *circunf* por 100 e guarda na variável *distância*. Pronto! Agora temos todos os dado e a linha 37 já pode calcular a velocidade. Veja que o valor calculado é armazenado na variável *veloc*.

Para visualizarmos a velocidade atingida pelo ímã, e portanto, da própria roda, usamos os comandos ***Serial.print****(xxxx)* e ***Serial.println****(xxxxx).* Note que nas linhas 39 e 40 usamos o ***Serial.print***, com isso podemos montar uma frase completa em apenas uma linha na tela do computador e na linha 41 temos o ***Serial****.println*, que ainda escreverá a informação na mesma linha que as escritas anteriores, mas o que vier depois passará a ser apresentado em uma linha abaixo. Sabendo disso, ficará fácil entender como a frase irá aparecer na tela do computador, observe:

Serial.print*(“Velocidade: “);*

Serial.print*(veloc);*

Serial.println*(“m/s”);*

Essas três linhas montaram a frase da seguinte maneira:

Figura 17 - *Tela do Monitor Serial mostrando a velocidade.*

Veja que foi passado a variável *veloc* para o ***Serial.print*** da linha 40, assim, toda vez que essa variável receber um novo valor, o computador nos mostrará em tempo real. Outro ponto a observar é que somente a variável não foi escrita entre aspas, a palavra *“Velocidade: ”* e *“**m/s**”* devem permanecer entre as aspas, isso deve ser feito com qualquer palavra ou frase que queira apresentar na tela sem que ela esteja vindo de uma variável.

Para que os dados possam ser visualizados no computador é necessário abrirmos o Monitor Serial da Arduino IDE e para fazer isso basta manter o Arduino conectado ao computador e clicar no ícone mostrado na figura abaixo:

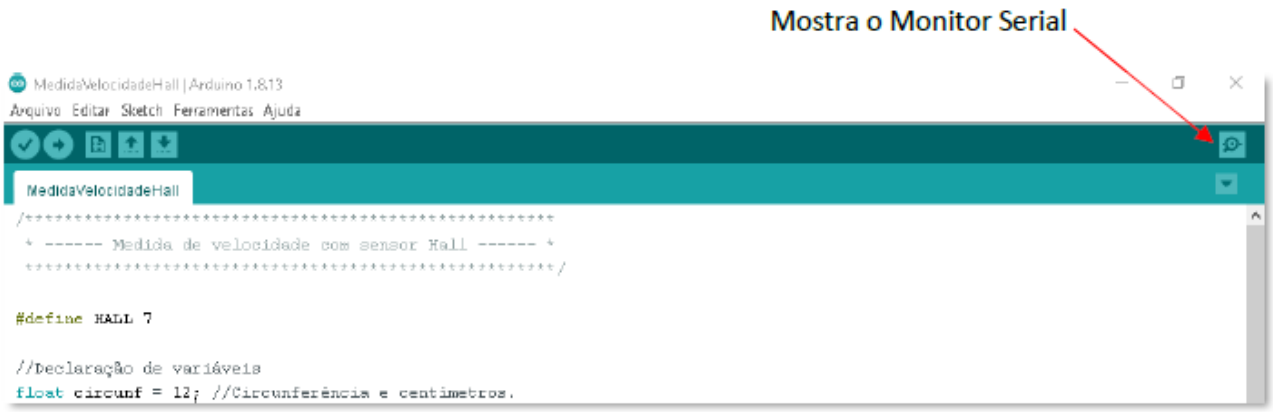


***Figura 18** - O ícone com a imagem de uma pequena lupa abre o Monitor Serial.*

Com toda essa explicação, entenda que a cada duas passagens do ímã em frente ao sensor é que o valor será calculado e apresentado na tela. A variável *ponto* é quem separa essas ações. Veja que dentro do bloco onde é registrada a primeira passagem do ímã, essa variável recebe o valor 1, o que impossibilita o programa de entrar novamente nesse bloco, mas poderá entrar no bloco seguinte com a segunda passagem do ímã, mas note que nesse segundo bloco, além de realizar o cálculo da velocidade, a variável *ponto* novamente muda de valor, passando a receber o valor 2 e isso não permite que esse bloco seja executado novamente em uma terceira passagem do ímã.

Para permitir que o programa volte a realizar um novo cálculo de velocidade, a variável *ponto* deverá receber o valor **zero** novamente e é justamente para isso que serve a função da linha 45, a qual observa se o ímã está longe do sensor e se a variável *ponto* já recebeu o valor **2**, caso esses critérios sejam verdadeiros, a variável *ponto* receberá o valor **zero**, o que fará com que um novo ciclo se inicie e o programa passe a refazer os cálculos para obter um novo valor de velocidade. A cada ciclo completo, uma linha com um valor de velocidade atual é acrescentada e mostrada na tela do Monitor Serial.

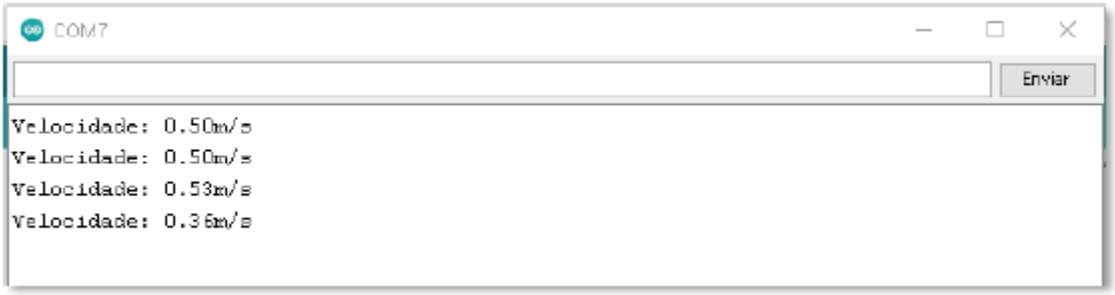


Figura 19 - *A cada ciclo do programa dentro do loop, um novo valor de velocidade é calculado e mostrado na tela do Monitor Serial.*

FUNCIONAMENTO

Com o Arduino conectado ao computador, abra o Monitor Serial. Logo de início nada estará sendo mostrado, mas ao passar o ímã duas vezes na frente do sensor, uma frase com o valor da velocidade deverá ser mostrado, conforme a figura 4. A cada duas passagens do ímã o Monitor Serial vai apresentando os resultados na tela, de acordo com a figura 6.

Uma sugestão de aplicação para esse projeto é o de estudar a velocidade de corpos, tanto retos como circulares. O valor da variável *circunf* poderá será alterado para a medida da distância entre dois ímãs presos, por exemplo, na lateral de um carrinho, sendo um na parte da frente e outro na parte de trás. Quando o carrinho passar em frente ao sensor, o primeiro ímã captura o tempo inicial e o segundo permite a realização do cálculo da velocidade, a qual será mostrada na tela do computador.

Figura 20 - *Ímãs presos na lateral de um carrinho.*

VOLTÍMETRO NO MONITOR SERIAL

Objetivo

Além de aplicar o que foi estudado em projetos anteriores, esse projeto tem o objetivo de apresentar a leitura de sinais analógicos pelo Arduino. Para isso, será construído um voltímetro que mostrará qual a tensão elétrica que um potenciômetro estará nos entregando ao passo que giramos seu cursor. A tensão será proveniente da própria placa Arduino.

Montagem do circuito

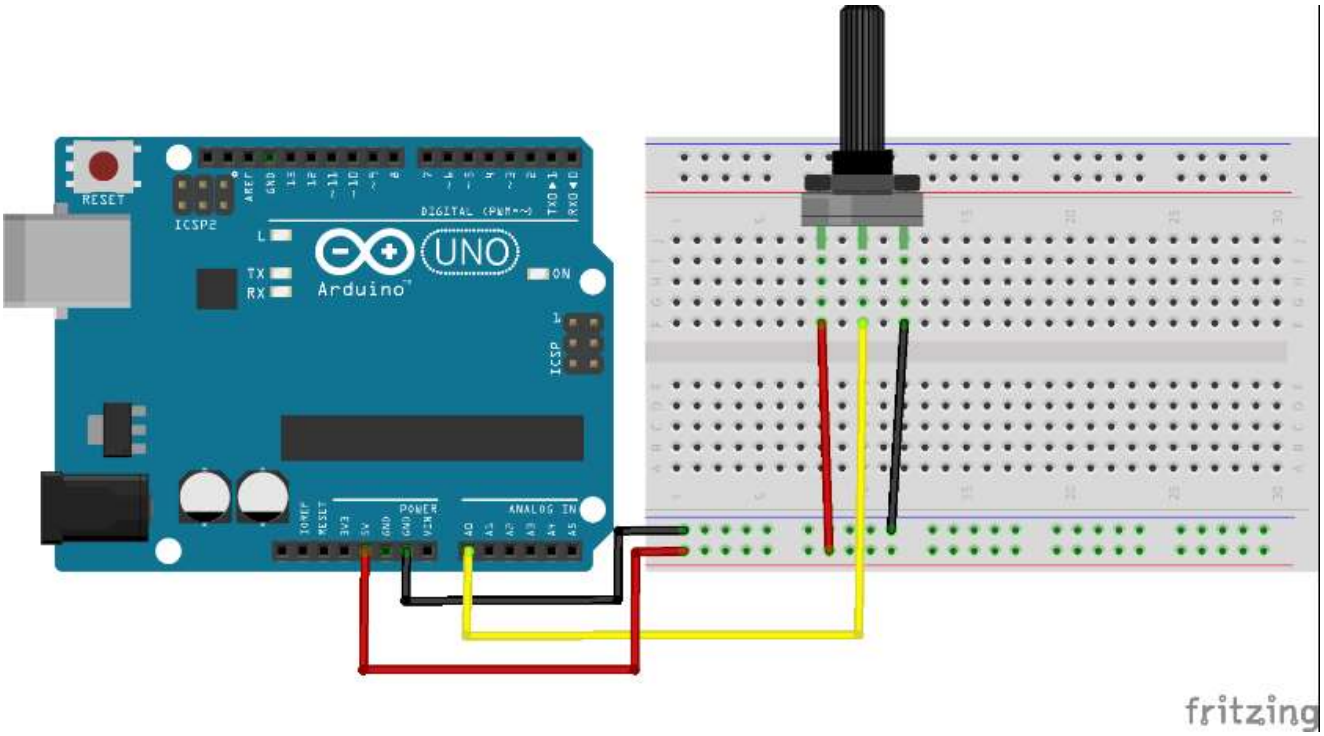


Figura 21 - *Circuito do voltímetro usando um potenciômetro para simular tensões diferentes de 0V a 5V.*

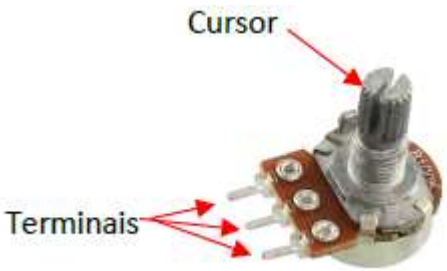


Figura 22 - *Aparência física de um potenciômetro.*

Potenciômetros não possuem polaridades definida entre seus pinos, o importante é saber que o terminal central deverá ser ligado ao pino A0 do Arduino e um dos terminais das extremidades ao 5V e o outro ao GND.

Lista de materiais

Arduino UNO
1 Potenciômetro de 5k, 10k ou 100k
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

A programação para esse projeto é bem simples. Veja:

```
/*********************************************
 * ------ Voltímetro no Monitor Serial ------ *
 *********************************************/

//Declaração de variáveis
float adc;
float tensao;

void setup() {

  Serial.begin(9600);

}

void loop() {

  adc = analogRead(A0);
  tensao = adc * 5.0/1023;

  Serial.print("Tensão: ");
  Serial.print(tensao);
  Serial.println("V");

}
```

EXPLICANDO O SKETCH

As declarações de variáveis e a configuração da comunicação serial já foram explicadas em projeto anterior, portanto, iremos nos ater apenas na explicação da leitura do sinal analógico, mas não iremos nos aprofundar no estudo desse tipo de sinal. Bem resumidamente podemos dizer que um sinal analógico é o tipo de sinal que possui uma larga faixa de valores. A grande maioria dos sinais que temos no mundo real, são analógicos, como por exemplo, a temperatura, os sinais vitais de um ser vivo, a umidade do ar, os sinais de carga e descarga de uma bateria, entre muitos outros tipos de sinais presentes na natureza.

O Arduino nos traz seis portas de entradas analógicas distintas, são elas a A0, A1, A2, A3, A4 e A5 o que significa que podemos fazer a leitura de seis sinais de naturezas diferentes. Nesse projeto usaremos a porta A0.

Observe na linha 17 como é simples fazer a leitura de um sinal analógico. A função *analogRead* é responsável por capturar o sinal presente na porta que está escrita entre seus parênteses, que em nosso caso é a porta A0. Note que o valor dela é repassado para a variável *adc*. Essa variável foi declarada como *float*, portanto, receberá valores com casas decimais.

Os valores entregue à variável pela função *analogRead* variam de 0 à 1023, porém, o valor que iremos medir irá variar de 0V à 5V, isso significa que a variável *adc* não receberá de imediato o valor que queremos. Para que isso seja possível devemos fazer uma simples regra de três, sabendo que 5V está para 1023, assim como 0 está para *x*, sendo *x* o valor que desejamos ler. Mas, não se preocupe com a formação desse cálculo agora, embora aconselho fortemente que estude sobre regras de três, pois, serão muito úteis na área da eletrônica. Veja na linha 18 que temos o cálculo para converter os valores conforme o que desejamos. Essa fórmula foi extraída de uma regra de três simples.

O valor lido na porta analógica e armazenado em *adc* é multiplicado pelo valor máximo de tensão que queremos, no caso os 5V e o resultado desse produto é dividido por 1023. Com isso a variável *tensao* irá receber o valor da tensão presente na porta analógica A0. As funções seriais das linhas 20, 21 e 22 montam a frase completa e permitem a exibição na tela do Monitor Serial, conforme já estudamos em outro projeto.

FUNCIONAMENTO

Conectando o Arduino ao computador e abrindo o Monitor serial, imediatamente serão mostradas várias linhas contendo o valor da tensão presente na porta analógica A0. A medida que alguém vá girando o cursor do potenciômetro, os valores na tela vão sendo alterados proporcionalmente. Girando o cursor em direção ao terminal GND, verá que a tensão vai caindo e girando no sentindo inverso, verá que a tensão vai subindo até o máximo de 5V.

A figura abaixo ilustra como os dados irão aparecer no Monitor Serial.

Figura 23 - *Monitor Serial mostrando a tensão presente na porta de entrada analógica A0.*

LDR PARA CONTROLE DE ILUMINAÇÃO

Objetivo

Com esse projeto, o aluno conhecerá o sensor de luminosidade LDR e usando os conhecimentos já adquiridos iremos implementar um circuito semelhante ao que existe nos postes da rede de distribuição pública de qualquer cidade. Consiste em um dispositivo que acenderá um LED quando o LDR passar a ficar no escuro, tal como a iluminação em uma cidade que se acende quando a noite chega. Com isso, firmaremos um pouco mais sobre o que estudamos a respeito das portas de entradas analógicas e começaremos a entender o uso de sensores analógico.

Montagem do circuito

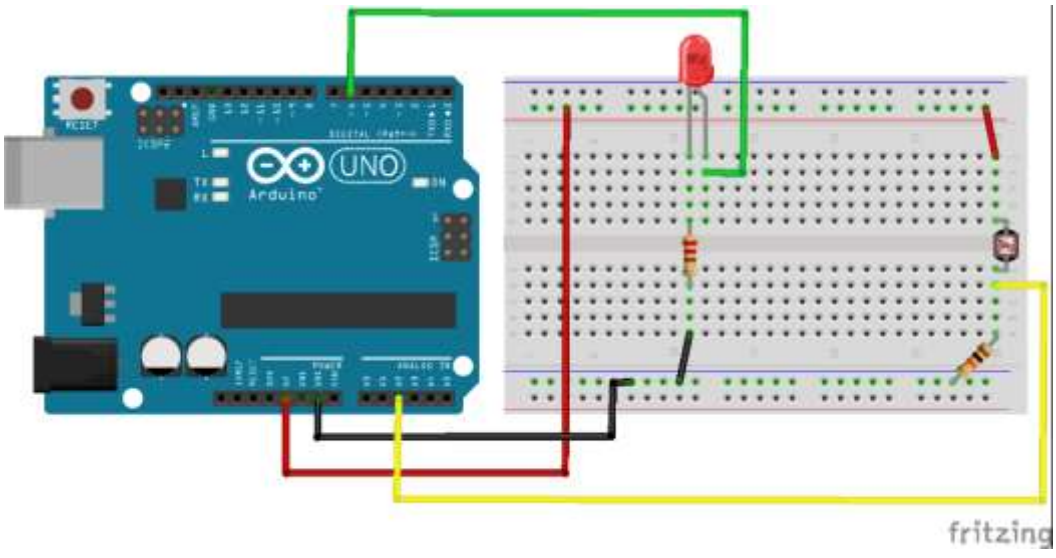


Figura 24 - *Circuito com LDR.*

Figura 25 - *Aparência física do LDR.*

O LDR é um tipo de resistor em que o valor de sua resistência depende unicamente da iluminação que recebe em sua parte sensível. A sigla LDR significa *Light Dependent Resistor*, traduzindo, Resistor Dependente de Luz. Não possui polaridade definida em seus terminais.

Lista de materiais

Arduino UNO
1 LED vermelho
1 Resistor 220Ω (vermelho, vermelho, marrom) para R1
1 Resistor 10k (marrom, preto, laranja) para R2
1 LDR
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

```
1  /*************************************************
2   * ------ LDR para controle de iluminação ------ *
3   *************************************************/
4
5  #define LUZ 6
6  #define LDR A2
7
8  //Declaraçãode variáveis
9  int adc;
10
11 void setup() {
12   Serial.begin(9600);
13   pinMode(LUZ, OUTPUT);
14
15 }
16
17 void loop() {
18
19   adc = analogRead(LDR);
20
21   Serial.print("ADC: ");
22   Serial.println(adc);
23
24   if(adc > 300)
25   {
26     digitalWrite(LUZ, LOW);
27   }
28   else
29   {
30     digitalWrite(LUZ, HIGH);
31   }
32
33 }
```

EXPLICANDO O SKETCH

Como podemos observar, não há nenhuma novidade nesse sketch, apenas fizemos um arranjo novo para explorar um pouco mais o que que estudamos sobre entrada analógica e comunicação serial. Note que foi feita uma definição para o pino A2 do Arduino e a ele foi atribuído o nome de LDR, isso apenas para que o aluno perceba que podemos definir nomes também para portas analógica da mesma forma como fizemos com as digitais.

Esse código funcionará da seguinte maneira, a variável *adc*, que foi declarada como do tipo *int*, receberá a leitura do sensor através da função ***analogRead***. Devemos aqui lembrar que essa função entrega para a variável, valores que vão de **0** à **1023** e nesse caso, podemos considerar zero para ausência de luz e 1023 para a máxima iluminação recebida pelo sensor. Não precisaremos realizar nenhum cálculo matemático de conversão, iremos usar essa faixa de valores para atuar sobre o LED.

Nas linhas 21 e 22, temos as funções que mostrarão na tela do Monitor Serial o valor que a variável *adc* estará recebendo em tempo real, assim facilita a escolha de um número para ser usado na comparação da linha 24. Nessa linha podemos ver que o *if* está comparando se o conteúdo de *adc* é maior que 300, se isso for verdade, veja na linha 26, a luz será apagada, pois LD1 receberá *LOW*. Mas,

se o valor de *adc* não for maior que 300, então a luz permanecerá acesa, já que LD1 estará recebendo *HIGH*, conforme podemos ver na linha 30.

O número 300 poderá ser substituído por qualquer valor que apareça no Monitor Serial e que acredite ser mais adequado para determinado tipo de iluminação.

FUNCIONAMENTO

Ligando o Arduino ao computador e abrindo o Monitor Serial, serão mostrados linha após linha os valores que o sensor LDR está entregando na entrada analógica A2. Esses valores dependerão muito da iluminação ambiente do momento. Passando a mão sobre o sensor, notará que o valor irá cair e a medida que o LDR vai sendo iluminado com a retirada da mão, os números irão aumentando proporcionalmente.

Caso o LED permaneça aceso com a iluminação existente no local, veja qual é o valor que está aparecendo nesse momento na tela do Monitor Serial e coloque um valor menor do que os 300 que usamos no sketch. Estando tudo certo, quando apagar a luz ambiente, o LED deverá acender e quando voltar a acender a luz ambiente o LED deverá apagar.

MEDINDO TEMPERATURA

Objetivo

Esse projeto apresentará uma maneira bem simples de realizar a medida de temperatura usando um sensor analógico, o LM35, muito fácil de usar e de programar. Um projeto perfeito para fixar os conhecimentos adquiridos nos projetos anteriores e aprender algo mais para complementar o aprendizado. O aluno poderá usar a ideia do que vai praticar aqui para elaborar sistemas de controle de temperatura ou alarmes que sinalizam quando a temperatura atinge certo valor.

Montagem do circuito

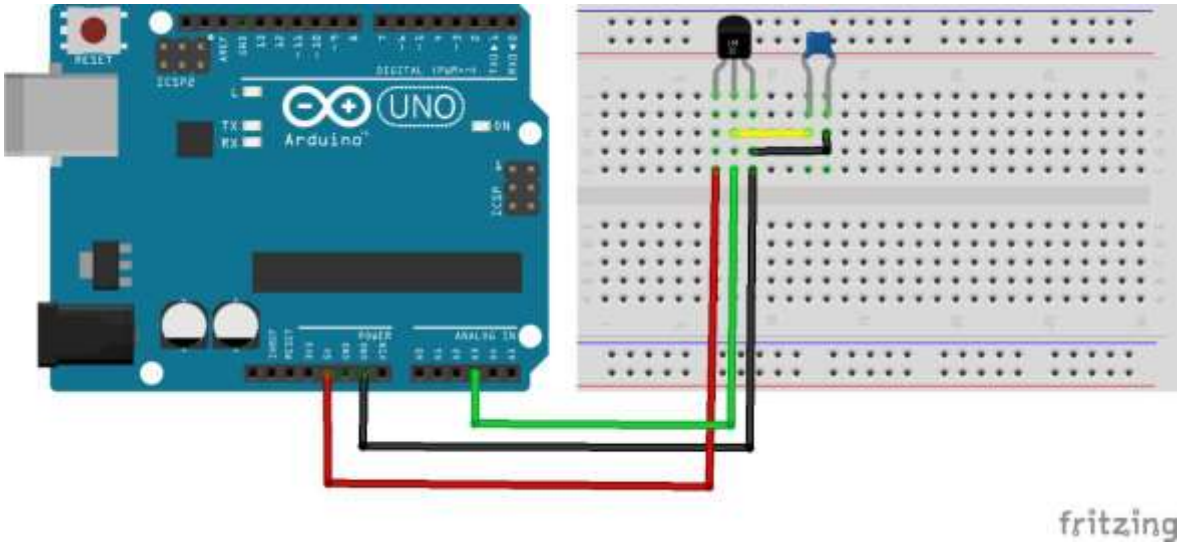


Figura 26 - *Circuito medidor de temperatura com o LM35 e o capacitor para filtrar do sinal.*

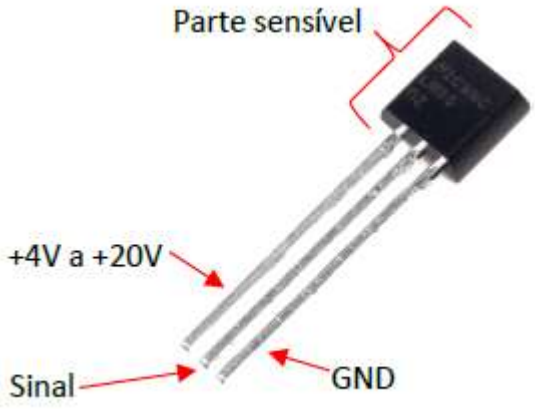


Figura 27 - *Sensor LM35.*

O LM35 é um sensor de temperatura que entrega em sua saída um valor de tensão elétrica proporcional à temperatura sentida em sua parte sensível. Essa proporção é de 10mV/°C, ou seja, 0,01V para cada 1°C.

É capaz de ler temperaturas que variam de -55°C à +150°C.

A saída de sinal é propícia a ruídos e para isso o fabricante recomenda o uso de uma capacitor para filtro entre o GND e o sinal.

Lista de materiais

Arduino UNO
1 Sensor de temperatura LM35
1 Capacitor cerâmico ou de poliéster de 100nF
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

A programação é bastante simples e utiliza elementos já estudado em projetos anteriores.

```
1  /*************************************
2   * ------ Medindo Temperatura ------ *
3   *************************************/
4
5  #define SENSOR A0
6
7  //Declaração de variáveis
8  float Temp;
9  int adc;
10
11 void setup() {
12
13   Serial.begin(9600);
14   analogReference(INTERNAL);
15
16 }
17
18 void loop() {
19
20   adc = analogRead(A0);
21
22   Temp = adc/9.3;
23
24   Serial.print("Temperatura: ");
25   Serial.print(Temp);
26   Serial.print("°C");
27   Serial.print("\t");
28   Serial.print("ADC: ");
29   Serial.println(adc);
30   delay(1000);
31
32 }
```

Explicando o sketch

Nesse projeto estamos usando a porta de entrada analógica A0, como se pode ver na linha 5. Todos os elementos de código pertencente à esse sketch já é de conhecimento do aluno, com exceção do que temos na linha 14, a função ***analogReference(INTERNAL).*** Para entender o porquê dessa função, o aluno precisa entender que a tensão padrão do Arduino UNO é de 5V e para realizar conversões de sinais analógicos para digitais, o microcontrolador usa essa tensão como referência e assim obtemos

a faixa de valores que vai de zero a 1023 bits. Com isso, temos que 5V está para 1023 bits, assim como zero volts está para zero bits.

O sensor de temperatura LM35, como explicado logo no início, entrega valores de tensões bem pequenos, mas, proporcionais à variação de temperatura e isso ocorre de forma linear. Sendo 0,01V para cada 1°C, quando a temperatura atingir 100°C, por exemplo, o sensor entregará 1V, ou seja, essa será a tensão máxima que o Arduino praticamente irá precisar para poder realizar a conversão com precisão de valores de temperatura que poderão variar de 0°C a 100°C.

Para mudar a tensão de referência e conseguirmos uma tensão menor, usamos a função ***analogReference(tipo)*** colocando como parâmetro o tipo de tensão de referência que precisamos para o projeto. A tabela seguinte mostra os tipos que podem ser usados com o Arduino UNO.

Tabela 1 - *Tipos de parâmetros e seus significados para a função analogReference(tipo).*

Tipo	Tensão elétrica
DEFAULT	5V
INTERNAL	1,1V
EXTERNAL	Depende da tensão aplicada no pino AREF.

Veja em nosso sketch, na linha 14, que estamos usando o parâmetro ***INTERNAL***, o que significa que o Arduino irá usar a tensão de 1,1V, existente internamente, como referência no momento da conversão dos valores, garantindo maior precisão na obtenção da temperatura. O tipo ***DEFAULT*** pode ser usado ou pode ser omitido. Se não usarmos a função ***analogReference(tipo)***, o Arduino irá usar os 5V como referência, portanto, usar o parâmetro *DEFAULT* ou não escrever a função ***analogReference(DEFAULT)*** dará no mesmo.

O parâmetro *EXTERNAL* fará com que o Arduino use o valor de tensão aplicado ao pino AREF da placa como referência. Essa tensão poderá ser proveniente de qualquer outra fonte de energia, exemplo, pilhas, baterias ou fontes de alimentação, mas essa tensão não poderá ultrapassar os 5V, pois, isso poderia causar danos irreversíveis à placa.

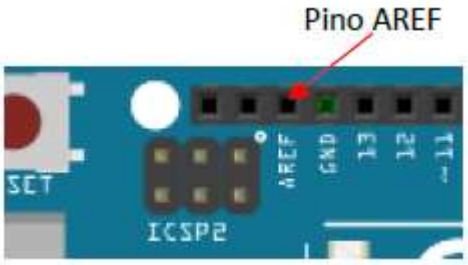


Figura 28 - *Localização do pino AREF no Arduino UNO.*

A linha 22 nos traz o cálculo de conversão. Como agora estamos usando 1,1V como tensão de referência, então podemos dizer que 1,1V está para 1023 bits, assim como zero volts está para zero bits e sabendo que 0,01V corresponde à 1°C, logo, 0,01V corresponde à 9,3 bits. Esse valor foi obtido por meio de uma simples regra de três. Observe:

1,1V ---------- 1023 bits

0,01V ---------- x bits

1,1x = 0,01 * 1023

1,1x = 10,23

x = 10,23 / 1,1

x = 9,3 bits

Pegando o valor que a variável *adc* estiver recebendo na linha 20 e dividindo por 9,3, obtemos o valor da temperatura.

Entre as linhas 24 e 29, temos as funções que imprimem na tela do Monitor Serial, o valor da temperatura juntamente com o valor da variável *adc,* apenas a título de estudos. O caractere *\t* da linha 27 serve apenas para dar espaço entre as letras. Logo abaixo dessas linhas, temos a função de tempo *delay* fazendo o código apresentar os valores a cada 1 segundo.

FUNCIONAMENTO

Ao ligar o Arduino ao computador e abrir o Monitor Serial, conseguirá ver linha-a-linha os valores de temperatura ambiente na tela e assim permanecerá enquanto a placa estiver ligada.

Como sugestão de exercício e usando o que já aprendeu, tente criar um código, baseado nesse, que seja capaz de acender um LED caso a temperatura ultrapasse um certo valor e desligue quando a temperatura cair abaixo de um determinado valor.

ROBÔ SEGUE LUZ

Objetivo

O principal objetivo desse projeto, é mostrar ao aluno como usar mais de um sensor analógico em diferentes portas de entrada analógica e fazê-los controlar dois motores de forma independente, ou seja, cada sensor controlará um determinado motor sem que o funcionamento de um interfira no outro. Esse projeto consiste em um robô que tem a capacidade de seguir um foco luminoso como o de uma lanterna, podendo seguir em frente, caso a fonte de luz ilumine os dois sensores com igual intensidade, ou fazendo curvas caso um sensor seja mais iluminado que o outro.

Será apresentado um circuito para acionamento dos motores, a conhecida ponte H, que nesse projeto usaremos de maneira simples, fazendo o motor seguir apenas para a frente e para os lados, mas não para trás.

Montagem do circuito

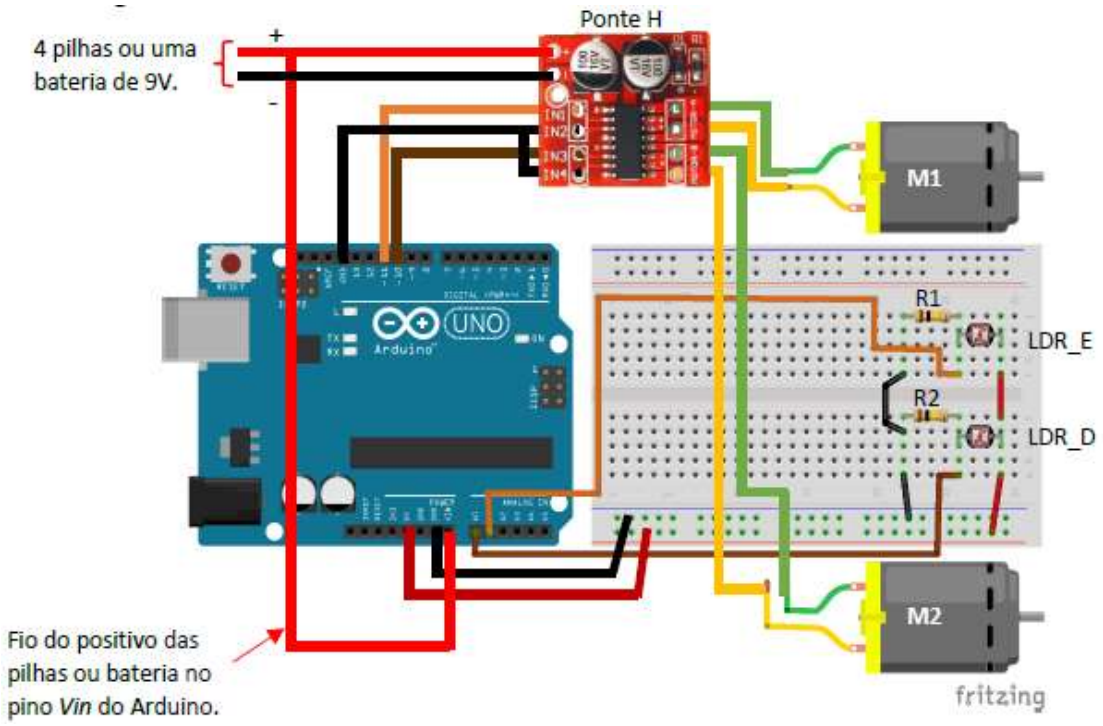


Figura 29 - *Circuito do robô Segue Luz com a mini ponte H.*

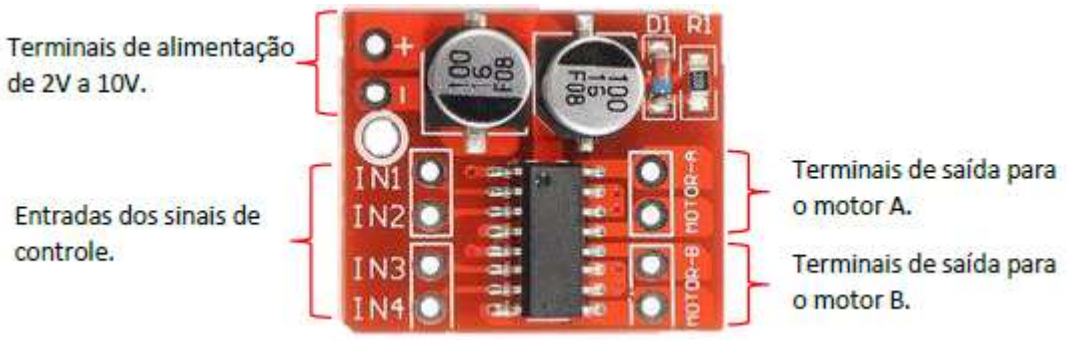


Figura 30 - *Mini ponte H com o circuito integrado L298N, para dois motores.*

A mini ponte H L298N tem capacidade de fornecer até 1,5A de corrente para cada um dos motores, sua alimentação pode ser de 2V até 10V e as entradas de sinais aceitam tensões que podem varias de 1,8V a 7V.

Lista de materiais

Arduino UNO
2 Motores de corrente contínua 3V a 9V
2 Resistores 100k (marrom, preto, amarelo) para R1 e R2
1 Mini Ponte H L298N
2 LDR
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

O código de programação para esse projeto está logo abaixo.

```
1  /**************************************
2   * -----------Robo segue luz-----------*
3   **************************************/
4  #define MOTOR1 11
5  #define MOTOR2 10
6  #define LDR_DIREITO A0
7  #define LDR_ESQUERDO A1
8
9  int LDR_D,
10     LDR_E,
11     Luz_Amb = 700;
12
13 void setup(){
14   Serial.begin(9600);
15   pinMode(MOTOR1, OUTPUT);
16   pinMode(MOTOR2, OUTPUT);
17 }
18
19 void loop() {
20
21   LDR_D = analogRead(LDR_DIREITO);
22   LDR_E = analogRead(LDR_ESQUERDO);
23
24   if((LDR_D == LDR_E) && (LDR_D > Luz_Amb))
25   {
26     digitalWrite(MOTOR1, HIGH);
27     digitalWrite(MOTOR2, HIGH);
28   }
29   else if(LDR_E > LDR_D && (LDR_E > Luz_Amb))
30   {
31     digitalWrite(MOTOR1, HIGH);
32     digitalWrite(MOTOR2, LOW);
33   }
34   else if(LDR_E < LDR_D && (LDR_D > Luz_Amb))
35   {
36     digitalWrite(MOTOR1, LOW);
37     digitalWrite(MOTOR2, HIGH);
38   }
39   else
40   {
41     digitalWrite(MOTOR1, LOW);
42     digitalWrite(MOTOR2, LOW);
43   }
44    Serial.print(LDR_E);
45    Serial.print("\t");
46    Serial.println(LDR_D);
47 }
```

EXPLICANDO O SKETCH

Como pode ver, esse código não traz nada de novidade, mas sim um conjunto de funções e comandos já estudados em projetos anteriores e com um arranjo bem interessante. Todas as definições, declarações de variáveis e configurações já são de conhecimento do aluno, portanto, vamos procurar entender o código de execução dentro de *loop()*.

As leituras dos dois sensores LDRs estão nas linhas 21 e 22. Seus valores são armazenados em tempo real nas variáveis *LDR_E* e *LDR_D*. Essas variáveis serão comparadas nos blocos da função *if* para que

seja tomada a decisão de qual motor mover de acordo com o sensor que receber mais luz, ou mover os dois ao mesmo tempo quando ambos os sensores receberam a mesma intensidade de iluminação.

A nossa função *if* está composta por quatro blocos de decisão. O primeiro bloco da linha 24, verifica se os dois sensores estão recebendo a mesma intensidade de luz e se o valor obtido pelo sensor *LDR_D* é maior que a luz ambiente. Caso todos esses critérios sejam atendidos, então os dois motores serão ligados e o robô deverá seguir em frente.

No bloco da linha 29 o programa analisa se o sensor *LDR_E* recebeu mais luz que o sensor *LDR_D* e se a intensidade luminosa recebida por *LDR_E* é maior que a luz ambiente. Em caso verdadeiro, o motor 1 liga e o motor 2 será desligado, com isso o robô deverá fazer a curva para a esquerda.

Chegando no bloco da linha 34 o programa verifica agora se o LDR_E está recebendo menor quantidade de luz em relação ao sensor LDR_D e se a intensidade luminosa de LDR_D é maior que a luz ambiente. Sendo verdadeiro todos esses critérios, então o motor 1 será desligado e o motor 2 será ligado, fazendo o robô virar para a direita.

Por fim, caso a luz recebida pelos dois sensores seja menor ou igual a luz ambiente, então os dois motores permanecerão parados, aguardando um foco de luz.

Para poder ajustar a variável *Luz_Amb*, a qual deverá conter o valor médio da luz ambiente de onde o robô esteja, temos as funções de saída serial entre as linhas 44 e 46, que mostrarão na tela do Monitor Serial, os valores recebidos por cada um dos sensores em luz ambiente. Não acenda lanternas ou outro foco de luz diretamente nos sensores para a verificação correta desses valores. Atribua como valor para a variável *Luz_Amb* um valor ligeiramente maior do que os valores capturados pelos sensores, exemplo, se o Monitor Serial estiver mostrando valores de aproximadamente 600, então coloque 700 para a variável.

FUNCIONAMENTO

Para testar o funcionamento, é necessário alimentar a ponte H com uma fonte de energia externa, a qual pode ser proveniente de 4 pilhas do tipo AA ou AAA, ou mesmo de uma bateria de 9V, isso vai depender do tipo de motor escolhido. Essa fonte de energia deverá ser ligada nos terminais de alimentação, conforme a figura 2, tomando o cuidado de obedecer a polaridade. Observe que um fio do terminal positivo dessa fonte deverá ser ligado no pino *Vin* do Arduino. Esse pino permite o circuito funcionar sem precisar estar conectado ao computador.

Uma dica importante, quando for gravar o código no Arduino e observar o valor para ajustar a variável *Luz_Amb*, mantenha a fonte de energia externa desligada do circuito. Depois que ajustar a variável, regravar o código com o novo valor para *Luz_Amb*, desconecte o Arduino do computador, ligue a fonte de alimentação e verifique o funcionamento. Quando apontar uma lanterna para os dois LDRs, o robô deverá se mover e seguir o foco de luz.

Ideia de motores:

Figura 31 - *Motores cc com redução.*

Ideia de montagem dos sensores:

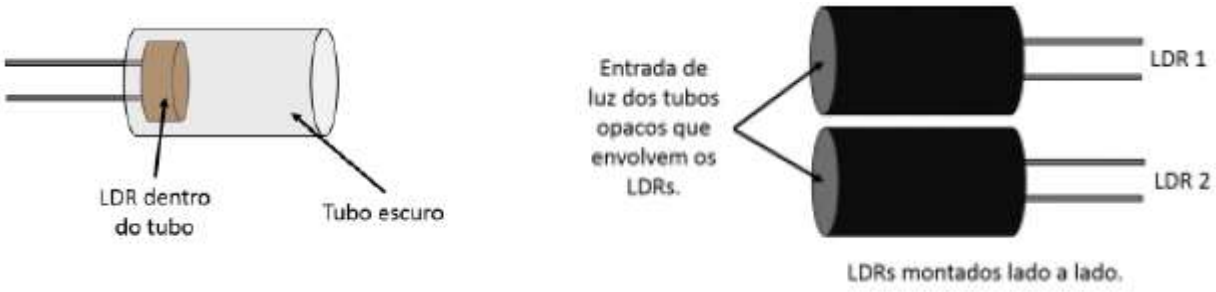


Figura 32 - *Montagem dos sensores em tubos opacos de preferência na cor preta.*

Ideia de uso:

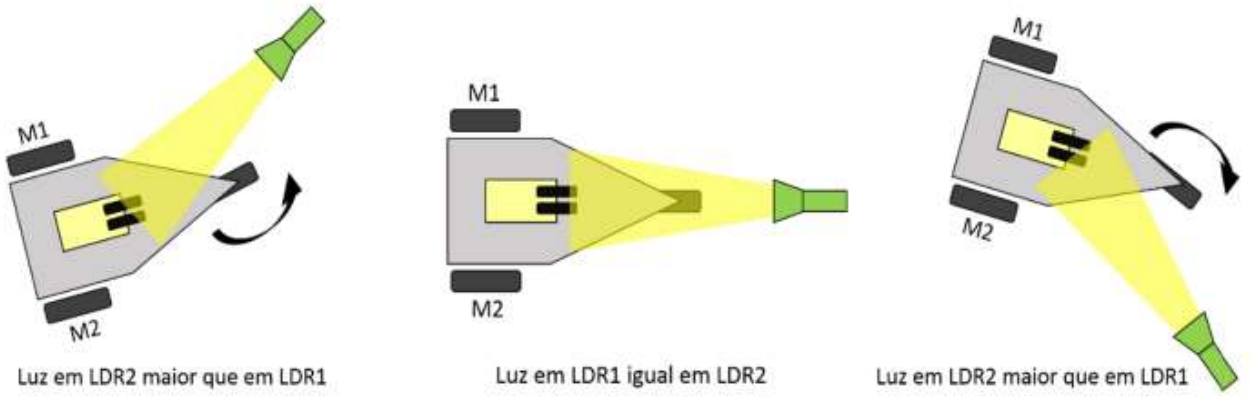


Figura 33 - *Modo de fazer o robô se mover.*

REGADOR DE PLANTAS AUTOMÁTICO

Objetivo

Nosso objetivo com este projeto, além de praticar mais sobre o uso da porta analógica, será o de realizar mais treinamentos com equações matemáticas na resolução de um problema físico, que neste caso será o de informar quando o solo, presente em um vaso com plantas, estiver com baixa umidade. É um projeto interessante e que traz uma excelente utilidade prática. E no que diz respeito à matemática, veremos como realizar a conversão de sinais digital em porcentagem.

Montagem do circuito

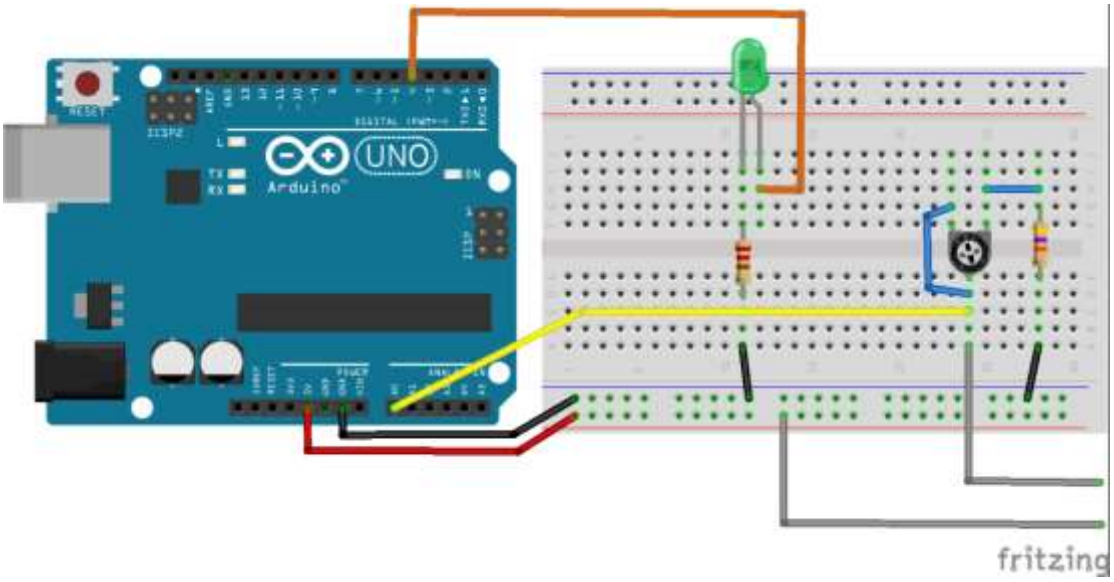


Figura 34 - *Circuito do regador de plantas automático.*

Na montagem desse circuito, os cabinhos X1 e X2, devem ser de pelo menos 20 cm.

O Trimpot – P1

Figura 35 - *Tipos de trimpots.*

Trimpots são resistores variáveis iguais aos potenciômetros, ambos com três terminais. A forma de ligar um potenciômetro é a mesma para um *trimpot*. São potenciômetros com tamanhos reduzidos que possuem a finalidade de permitir ajustes durante os testes de um circuito e após terem encontrado o ponto ideal do ajuste o gabinete de um determinado equipamento, em que se encontra tal circuito eletrônico, é fechado com o *trimpot* dentro impedindo que pessoas leigas girem seu cursor garantindo, com isso, a permanência do ajuste realizado.

Lista de materiais

Arduino UNO
1 Trimpot de 47kΩ para P1
1 Resistor de 220Ω (vermelho, vermelho, marrom) para R1
1 Resistor de 47kΩ (amarelo, violeta, laranja) para R2
1 LED verde, amarelo ou vermelho
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

```
/***********************************
 * ------ Regador de plantas ------ *
 ***********************************/


float adc;
float umidade;
float compare = 0.00;
float setPoint = 25.00;
float histerese = 20.00;

void setup() {

  Serial.begin(9600);
  pinMode(4, OUTPUT);
  digitalWrite(4, LOW);

}

void loop() {

  adc = analogRead(A0);

  umidade = 100.00*adc/1023.00;

  if(compare != umidade)
  {
    Serial.print(umidade);
    Serial.println("%");

    if(umidade < setPoint)
    {
      digitalWrite(4, HIGH);
    }
    else if(umidade >= (setPoint + histerese))
    {
      digitalWrite(4, LOW);
    }
    compare = umidade;
    delay(1000);
  }

}
```

EXPLICANDO O SKETCH

O código apresentado para este projeto é bastante simples, nenhuma novidade em termos de programação ou configuração estão presentes, o que temos aqui, logo de início são cinco variáveis do tipo *float* que foram declaradas como globais e as três últimas foram inicializadas com valores pré-definidos. Na função *setup()* temos a inicialização da comunicação serial com taxa de transmissão em ***9600 bps*** e nas linhas abaixo, podemos ver a escolha do pino 4 do Arduino para ser uma saída de dados digital e na linha 16 estamos inicializando esse pino em nível lógico baixo, garantindo que o LED inicialize desligado.

Como o objetivo do circuito é detectar o percentual de umidade existente em uma porção de solo, então podemos deduzir que a leitura será de um sinal analógico, logo, escolhemos para isto o pino *A0*. Na linha 22, já dentro da função *loop()*, encontramos a variável *adc* recebendo os valores resultantes da leitura analógica pelo pino *A0*. Como já foi explicado anteriormente, essa variável irá armazenar valores de ***zero*** a ***1023***, mas, como estamos usando uma variável do tipo *float*, logo o mais correto é dizer que serão armazenados valores de ***0.00*** a ***1023.00*****.**

Os valores que serão apresentados no Monitor Serial e também usados para realizar a comparação para o comando do LED, deverão ser em porcentagem e para isso devemos converter a faixa de valores que vai de ***0.00*** a ***1023.00*** na faixa que irá de ***0.00*** a ***100.00*** e para isto temos a equação matemática da linha 24. Observe que o resultado dessa equação será armazenado na variável *umidade*. Uma observação interessante a se fazer é que nesse tipo de equação, o número ***100.00*** pode ser substituído pelo maior número de uma faixa qualquer, por exemplo, digamos que seja necessário converter de ***0.00*** a ***1023.00*** para ***0.00*** a ***255.00***, então é só trocar o número ***100.00*** pelo número ***255.00***. Caso não se lembre, essa mesma fórmula foi usada no projeto "*VOLTÍMETRO NO MONITOR SERIAL*", onde o valor máximo escolhido foi ***5.0***.

Na linha 26 temos a função *if* verificando se a variável *compare*, que inicialmente foi inicializada com o valor *zero*, é diferente "*!=*" do valor contido na variável *umidade*. Caso seja, as linhas 28 e 29 mostrarão no Monitor Serial o valor na tela em porcentagem. Na linha 31, passamos para um outro *if* que agora verifica se o valor da variável *umidade* é menor que o valor da variável *setPoint*, a qual, recebeu inicialmente, na linha 9, o valor *25.00*. Se o valor contido em *umidade* for menor que *25.00*, então, o programa irá executar o comando da linha 33, enviando *HIGH* para o pino 4 e, com isso, acender o LED. No entanto, se o valor de *umidade* for maior que o resultado da soma entre as variáveis *setPoint* e *histerese*, o que pode ser visto na linha 35, então o programa passará a executar o comando da linha 37, que consiste em enviar nível lógico baixo para o pino 4 fazendo o LED ser desligado. A variável *histerese* serve como uma folga para que o circuito não fique ligando e desligando a carga (LED) em intervalos muito curtos.

Na linha 39, a variável *compare* está recebendo o valor atual da variável *umidade*, com isso, todo o processamento explicado só irá ocorrer novamente caso ocorra uma leitura diferente de valores pela porta analógica. E para finalizar, na linha 40 temos a função ***delay()*** garantindo uma pausa de ***1 segundo*** **(*1000 milissegundos*)** antes de o programa realizar nova leitura.

FUNCIONAMENTO

Após carregar o código para o Arduino, abra o Monitor Serial e verifique os valores aparecendo. Use os cabinhos X1 e X2, mostrados na figura 1, como sensores de umidade e para que eles realizem a leitura do solo, espete as pontas de cada um na terra contida em um pequeno vazo com plantas, nesse momento, é importante que a terra esteja seca. Mantenha uma distância de até 2 cm entre X1 e X2. Feito isto, realize o ajuste em P1. Como a terra estará sem umidade, vá ajustando P1 até que no Monitor Serial seja mostrado o valor 0.00%, ou muito próximo disso. Vá acrescentando a quantidade de água necessária para a planta e olhe os resultados na tela do computador. O valor da variável *setPoint* pode ser substituído por valores que se adéquem ao tipo de planta ou quantidade de umidade que se queira manter no solo. A variável *histerese* também pode ser alterada conforme a necessidade.

Um ponto interessante a ser observado é que ocorrerá eletrólise entre X1, X2 e a água, isto causará oxidação em X2 por estar ligado ao polo positivo, ou seja, se ficar no solo úmido por vários dias, a parte metálica será consumida e terá que ser substituída. Uma ideia seria usar fio inoxidável.

EFEITO CHAMAS COM LEDS

Objetivo

Nesse projeto, o aluno vai aprender a usar a função PWM existente em algumas portas digitais do Arduino. Conhecerá também uma função que gera números aleatórios, o que vai ser perfeito para provocar o efeito luminoso esperado. O projeto consiste em um circuito que provocará um efeito visual interessante que lembrará as chamas, podendo ser usado para compor cenários, substituir velas em eventos ou para simular uma lareira. Tudo dependerá da criatividade e necessidade do aluno.

Montagem do circuito

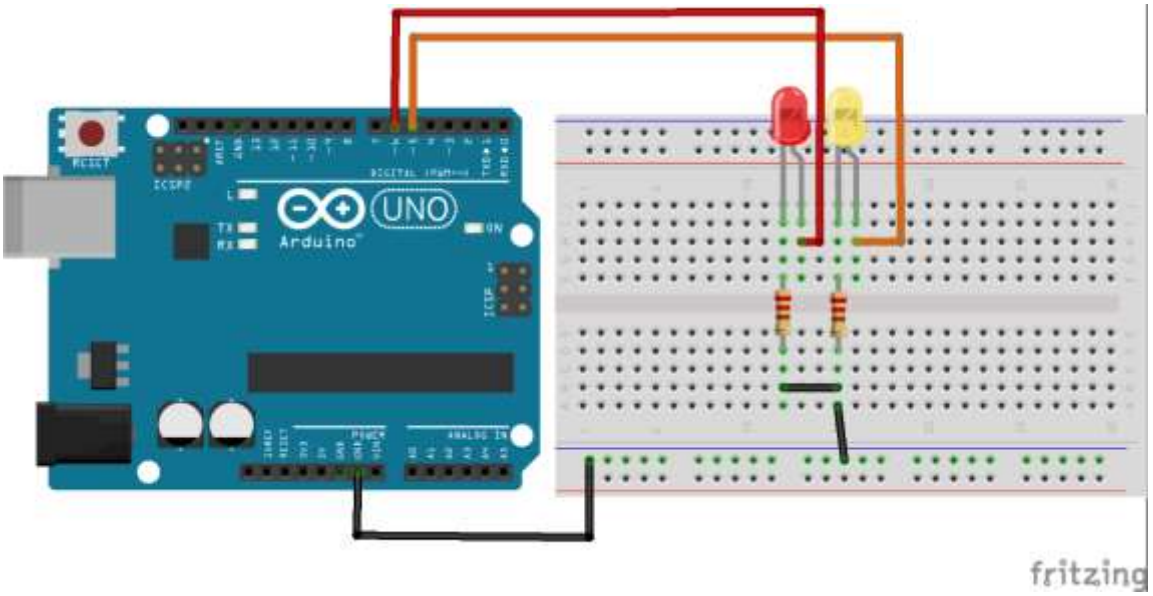


Figura 36 - *Circuito que provocará o efeito chamas.*

Lista de materiais

Arduino UNO
1 LED vermelho (preferencialmente de alto brilho)
1 LED amarelo (preferencialmente de alto brilho)
2 Resistor 220Ω (vermelho, vermelho, marrom)
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

Escreva o código mostrado abaixo na Arduino IDE e envie para a placa.

```
1  /**************************************
2   * ------ Efeito chamas com LEDs ------ *
3   **************************************/
4
5  #define LED_Vm 5
6  #define LED_Am 6
7
8  int Vm,
9      Am;
10
11 void setup() {
12
13   randomSeed(analogRead(A0));
14 }
15
16 void loop() {
17
18   Vm = random(50, 256);
19   Am = random(50, 256);
20
21   analogWrite(LED_Vm, Vm);
22   analogWrite(LED_Am, Am);
23
24   delay(100);
25
26 }
```

Explicando o sketch

Apesar de ser um programa bem simples, ele traz dois elementos muito úteis no desenvolvimento de diversos projetos e vamos começar entendendo o que temos na função *setup()*.

Para criarmos um efeito semelhante ao cintilar de chamas, precisamos fazer com que os LEDs, sendo um vermelho e outro amarelo, modifiquem a intensidade de seus brilhos de forma aleatória, sem seguir um padrão definido e a maneira mais simples de realizar esse feito é através do uso da função *random(valor).* Na linha 13 podemos ver como configurar essa função.

Quando chamamos a função ***analogRead(porta analógica)*** para fazer a leitura de qualquer porta de entrada analógica e não conectamos fisicamente ela a nada, então, ela começa a capturar ruídos aleatórios, provenientes de interferências no ambiente e até mesmo no próprio circuito eletrônico. A função ***randomSeed(analogRead(A0))***, na linha 13, configura *random* para usar como fonte de sinal aleatório o canal analógico *A0*. Com isso podemos obter valores aleatórios que podem varias de 0 a 1023.

A função random pode receber um único valor inteiro, sendo no máximo de até 2.147.483.647, e assim, os números serão sorteados de zero até o valor especificado menos um. Exemplo**:** ***random(500)***, irá gerar valores aleatórios entre zero e 499. Essa função também poderá receber dois

parâmetro, onde o primeiro corresponde ao valor mínimo e o segundo ao valor máximo e assim os valores sorteados ficarão entre o mínimo e o máximo menos um. Exemplo: ***random(10, 125)*** nos dará valores aleatórios entre 10 e 124. Esse é o tipo que estamos usando nesse projeto.

Nas linhas 18 e 19 temos as variáveis *Vm* e *Am* recebendo valores aleatórios de modo independentes uma da outra. As funções *random* em cada uma têm como parâmetro mínimo o valor

50 e como máximo 256. Com isso, as variáveis receberão constantemente valores aleatórios que estarão compreendidos entre esse intervalo.

Até o momento, fizemos diversos projetos que faziam LEDs ficarem acesos ou apagados, mas nunca com baixo ou meio brilho. Isso porque as saídas digitais trabalham enviando níveis lógicos baixo ou alto, conforme o que vimos em projetos anteriores. No entanto, para fazer um efeito semelhante à chamas, não podemos fazer os LEDs ficarem acendendo e apagando, não ficaria nada parecido com fogo. Para que esse efeito funcione bem, os LEDs deveram alterar a intensidade de seus brilhos entre um valor mínimo e máximo. Para isso, acionamos os LEDs com sinal PWM.

Não iremos aqui entrar em detalhes sobre o PWM para não estender muito o assunto, mas entenda que esse tipo de sinal simula um sinal analógico, fazendo um pino digital variar de zero ao máximo de 255. PWM (*Pulse Width Modulation*) que significa Modulação por Largura de Pulso está presente em seis portas digitais do Arduino UNO, são elas, as portas 3, 6, 5, 9, 10 e 11, representadas por um ~ desenhado na placa ao lado do pino. Esses pinos de saídas digitais, podem simular um sinal analógico por meio da função ***analogWrite(pino, valor)*** e esse sinal pode tanto variar o brilho de LEDs como também controlar a velocidade em motores de corrente contínua ou o aquecimento de uma estufa.

Nas linhas 21 e 22 temos a função ***analogWrite(pino, valor)*** sendo aplicada em cada LED. Para exemplo tomemos a situação do LED vermelho:

analogWrite(LED_Vm, Vm);

Onde, ***LED_Vm*** corresponde ao pino 5, definido na linha 5 do código e *Vm* é a variável que está recebendo valores da função ***random(50, 256)***, na linha 18 e que fará o PWM variar entre 50 e 255. O mesmo ocorre para o LED amarelo.

Funcionamento

Após carregar o código no Arduino, os LEDs começaram a se comportar como uma chama, variando seu brilho aleatoriamente. E para que o efeito fique mais interessante, envolva os LEDs com uma folha

de papel branca de modo a escondê-los e com isso teremos a mistura das luzes sendo irradiada através da folha, dando a impressão de que há fogo.

CONTROLANDO BRILHO DE LED POR POTENCIÔMETRO

Objetivo

Nesse projeto iremos explorar um pouco mais a aplicação do sinal PWM e para ficar mais interessante, vamos fazer com que um sinal analógico controle os níveis PWM e para que isso seja possível será apresentada uma função bastante útil que realiza regra de três de forma bastante simples, convertendo uma determinada faixa de valores em outra e assim adequando valores.

Montagem do circuito

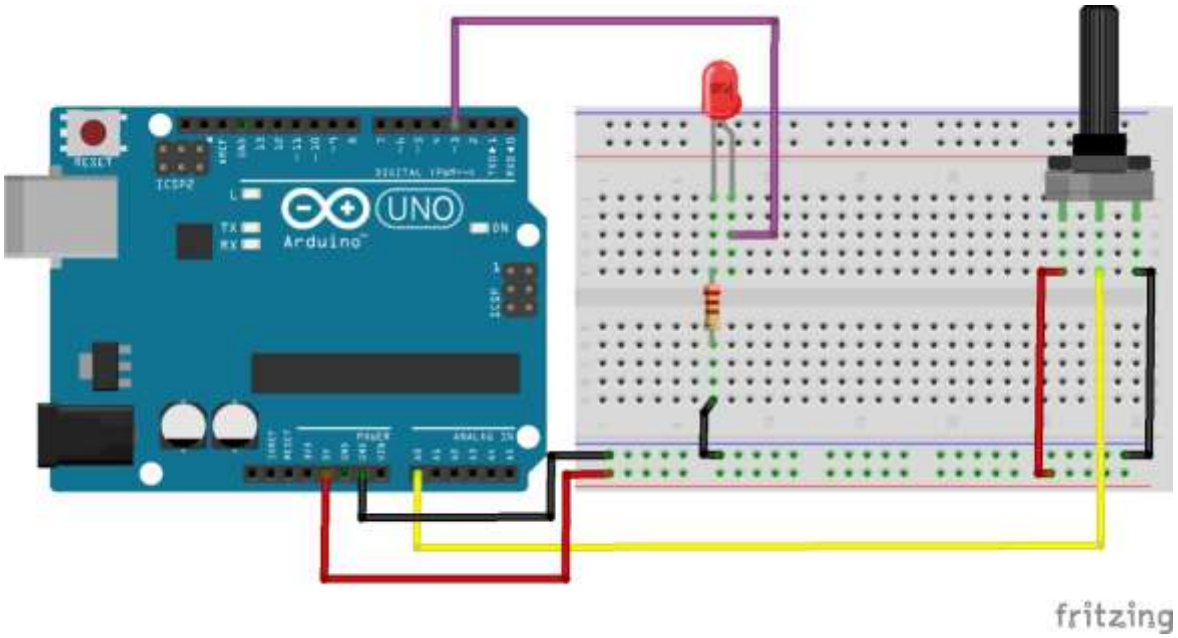


Figura 37 - *Controle de LED com potenciômetro.*

Lista de materiais

Arduino UNO
1 LED vermelho ou qualquer outra cor
1 Resistor 220Ω (vermelho, vermelho, marrom) para R1
1 Potenciômetro 5k ou 10k para P1
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

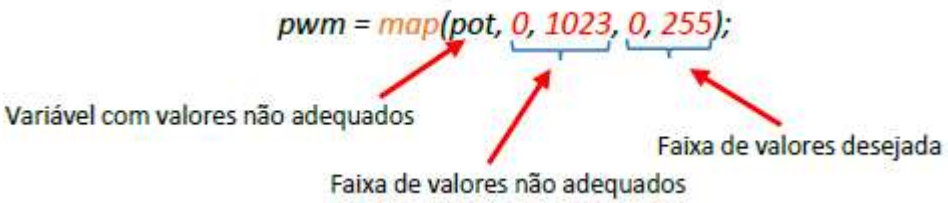
Temos aqui um sketch bem simples. Escreva-o na Arduino IDE e envie para a placa.

```
1  /****************************************
2   * ------ Efeito chamas com LEDs ------ *
3   ****************************************/
4
5  #define LED 3
6
7  int pot;
8  int pwm;
9  int compare = 0;
10
11 void setup() {
12
13   Serial.begin(9600);
14
15 }
16
17 void loop() {
18
19   pot = analogRead(A0);
20
21   pwm = map(pot, 0, 1023, 0, 255);
22
23   analogWrite(LED, pwm);
24
25   if(compare != pwm)
26   {
27     Serial.println(pwm);
28     compare = pwm;
29   }
30
31 }
```

Explicando o sketch

Esse sketch nos permite controlar os níveis do sinal PWM por meio de uma das portas de entrada analógica do Arduino. Para que isso seja possível criamos duas variáveis, uma para receber os sinais analógicos produzidos pelo potenciômetro (P1) e outra para receber os níveis PWM. Como pode-se perceber olhando na linha 19, a variável *pot* recebe os níveis de sinais analógicos de A0. O aluno deve sempre ter em mente que tais níveis possui intervalo de 0 a 1023 e esses serão os valores armazenado pela variável, dependendo da posição do cursor de P1. No entanto, sabemos também que os níveis de sinal PWM possui um intervalo menor, variando de 0 a 255, sendo assim, os sinais analógicos só conseguiriam controlar a saída PWM dentro dessa faixa de valores, mas quando ultrapassar, haverá uma instabilidade no funcionamento e o PWM poderá assumir qualquer valor indesejado.

Para contornar o problema da diferença de níveis de sinais, usaremos a função *map()*. Essa função transforma uma determinada faixa de valores, provenientes de uma variável, em uma faixa adequada para um projeto. Na linha 21 estamos fazendo a variável *pwm* receber a faixa de valor normatizada pela função ***map()***. Vamos entender:



Com isso, temos uma regra de três pronta, fazendo **zero** estar para **zero**, assim como **1023** está para **255** e é dessa forma que conseguiremos controlar o brilho de um LED, ou a velocidade de um motor usando os níveis de sinais analógicos sem nenhum problema.

Na linha 23 temos a função PWM que envia os níveis de **0** a **255**, normatizados pela função ***map()*** e armazenados na variável *pwm*, para controlar o LED.

O bloco *if* da linha 25 verifica se o valor de *pwm* foi modificado comparando essa variável com o valor armazenado em *compare*. Caso os valores de *pwm* e de *compare* sejam diferentes, o Monitor Serial irá nos mostrar o valor de *pwm* e na linha 28 o programa faz *compare* receber o mesmo valor que *pwm*. Essa comparação é interessante para que a saída de comunicação serial não fique enviando constantemente valores repetidos, mas somente quando ocorrer uma mudança.

Funcionamento

Depois que o Arduino estiver conectado ao computador, mova o cursor do potenciômetro e verifique o brilho do LED, esse deve variar proporcionalmente com a posição do cursor. Para visualizar os níveis de PWM que o LED está recebendo, abra o Monitor Serial e movimente o cursor do potenciômetro. Os valores na tela irão ficar parados toda vez que o potenciômetro estiver sem movimento e mostrará um novo valor quando o cursor for modificado de posição. Pode acontecer de os valores na tela não pararem de se movimentar, isso ocorrerá caso a posição do cursor esteja entre o limiar de dois valores, ou por interferências mecânicas do potenciômetro, ou mesmo interferências captadas pelo jumper do pino A0.

CONTROLE DE VELOCIDADE DE MOTOR POR LAÇO DE REPETIÇÃO

Objetivo

Com esse projeto será possível mostrar de forma prática o controle de velocidade de um motor de corrente contínua, mas, será um controle autônomo que fará a velocidade do motor aumentar e diminuir sozinha, ou seja, sem a intervenção humana. A parte interessante nesse projeto é o uso de um transistor NPN como driver para o motor.

Montagem do circuito

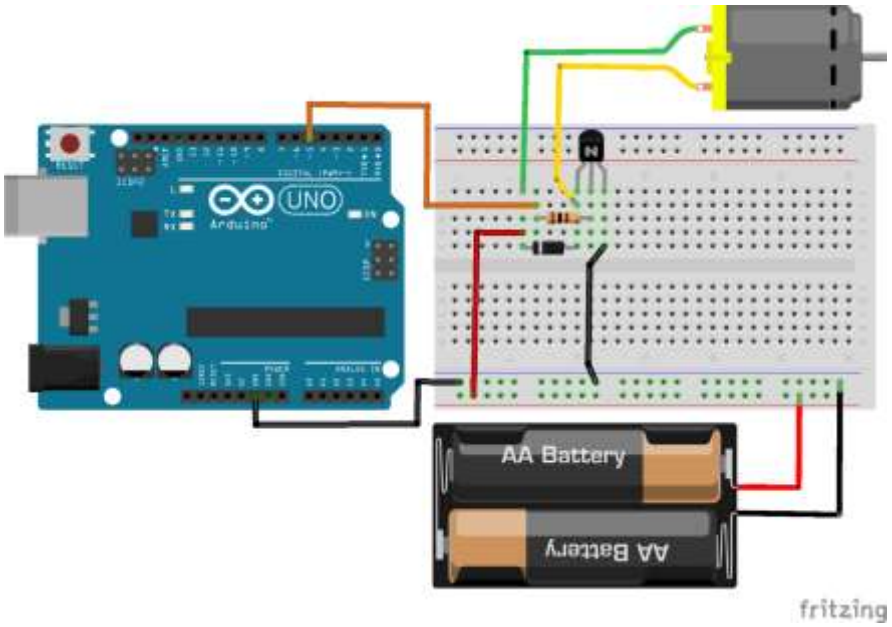


Figura 38 - *Circuito de controle do motor.*

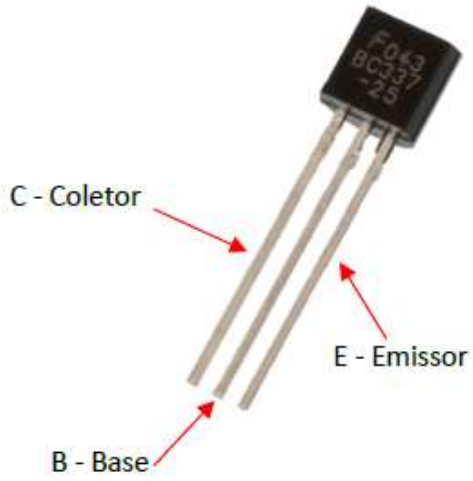


Figura 39 - *Transistor NPN - BC337*

O transistor BC337, usado nesse projeto, é do tipo NPN, o que significa que possui a capacidade de controlar cargas negativas entre coletor e emissor quando recebe sinal positivo em sua base. Pode controlar cargas de até 800mA, o que o torna suficiente para o nosso projeto. O uso do transistor evita

sobrecarregar os pinos digitais do Arduino, sendo que esses podem fornecer no máximo 40mA e um motor básico pode chegar a consumir 500mA ou mais, o que poderia queimar o chip ATMEGA328P caso o motor fosse ligado direto.

Sobre o diodo D1, no momento da montagem, verifique a posição do terminal catodo, representado pela faixa branca, ela deve estar na mesma posição mostrada na figura 1. Esse diodo protege o transistor contra a F.C.E.M (Força Contra Eletromotriz) produzida pelo motor quando ele é desligado.

Lista de materiais

Arduino UNO
1 Motores de corrente contínua 3V
1 Resistor 1k (marrom, preto, vermelho) para R1
1 Transistor BC337 ou BC548
1 Diodo 1N4007 ou 1N4004
2 Pilhas AA
1 Suporte para duas pilhas AA
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

```
/*************************************************
 * ------ Velocidade de motor com laço for ------ *
 *************************************************/

#define MOTOR 5

void setup() {


}

void loop() {

  for(int i = 0; i < 256; i++)
  {
    analogWrite(MOTOR, i);
    delay(50);
  }

  for(int i = 255; i > 0; i--)
  {
    analogWrite(MOTOR, i);
    delay(50);
  }

}
```

Explicando o sketch

O sketch desse projeto é bastante simples, veja que não foi necessário usar a função *setup()*, a qual permaneceu vazia. Mas, a função *loop()* nos traz elementos de programação que já apresentamos em projetos anteriores e nenhuma novidade a mais.

Na linha 14 temos um bloco do laço de repetição *for*, que vai incrementando a variável local *i*, que se inicia em **zero**, até **256**. Esses valores de *i* vão compondo o sinal PWM para a função ***analogWrite*** que os envia para o pino 5 do Arduino, definido na linha 5 como MOTOR. A cada passo de 50 milissegundos, tempo esse determinado pela função ***delay(50),*** a velocidade do motor vai aumentando até que *i* atinja o valor de **255**, limite máximo do PWM, com isso o motor estará em sua máxima velocidade e o programa sai desse laço de repetição.

Assim que o motor atingir a velocidade máxima, o programa passa para o próximo bloco de *for* na linha 20. Esse bloco fará o inverso do primeiro. Perceba que *i* inicia-se agora em **255**, máximo valor do PWM, e a cada passo de 50 milissegundos, vai decrementando até chegar em **zero**, mínimo valor PWM. Com isso, note que a velocidade do motor vai diminuindo graças a função ***analogWrite*** que ficará enviando os valores de *i* para o pino 5 do Arduino. Quando o PWM chegar em zero, o programa volta para o primeiro laço *for* e começa tudo de novo.

Funcionamento

Assim que o programa for gravado no Arduino perceberá que o motor começará a girar lentamente, mas, à medida que o tempo passa sua velocidade vai aumentando até chegar ao seu limite máximo. Quando chegar a atingir o máximo, a velocidade começa a diminuir até o valor mínimo e, após isso, volta a aumentar novamente. Esse comportamento de aumentar e diminuir a velocidade constantemente permanecerá ocorrendo enquanto o Arduino estiver ligado.

Como forma de exercício, tende fazer o Arduino mostrar no Monitor Serial os valores PWM que o motor está recebendo.

CONTROLE DE VELOCIDADE DE MOTOR POR BOTÕES

Objetivo

Nesse projeto, o aluno aprenderá a efetuar o controle de velocidade de um motor de corrente contínua usando dois botões para aumentar e diminuir o valor do PWM. O circuito é praticamente o mesmo usado no projeto anterior, mas com o acréscimo de dois botões táteis. Na programação verá que não existe nada novo, apenas um arranjo adequado para o perfeito funcionamento desse circuito.

Montagem do circuito

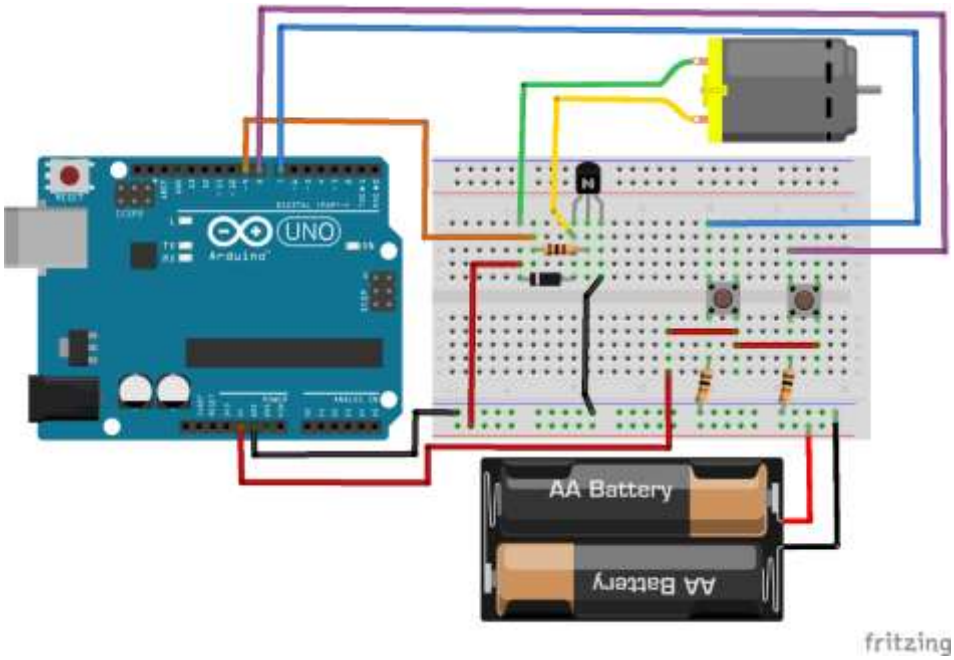


Figura 40 - *Montagem do circuito de controle de velocidade.*

Lista de materiais

Arduino UNO
1 Motores de corrente contínua 3V
1 Resistor 1k (marrom, preto, vermelho) para R1
2 Resistores 10k (marrom, preto, laranja) para R2 e R3
1 Transistor BC337 ou BC548
2 Botões táteis tipo Push-Pull
1 Diodo 1N4007 ou 1N4004
2 Pilhas AA
1 Suporte para duas pilhas AA
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

```
1  /*************************************************************
2   * ------ Controle de velocidade de motor por botões ------ *
3   *************************************************************/
4
5  #define MOTOR 9
6  #define BTN_1 7
7  #define BTN_2 8
8
9  int btnMais,
10     btnMenos,
11     pwm = 0;
12
13
14 void setup() {
15
16   pinMode(BTN_1, INPUT);
17   pinMode(BTN_2, INPUT);
18
19 }
20
21 void loop() {
22
23   btnMenos  = digitalRead(BTN_1);
24   btnMais   = digitalRead(BTN_2);
25
26   if(btnMais)
27   {
28     pwm += 5;
29     if(pwm > 255) pwm = 255;
30     delay(100);
31   }
32   else if(btnMenos)
33   {
34     pwm -= 5;
35     if(pwm < 0) pwm = 0;
36     delay(100);
37   }
38
39   analogWrite(MOTOR, pwm);
40
41 }
```

Explicando o sketch

Para esse sketch, foram criadas três variáveis do tipo int, sendo duas para receber os dados dos botões e uma para aplicar o PWM, esta última foi inicializada com o valor zero para garantir que o pino 9, o qual definimos com o nome de MOTOR, sempre inicie desligado. Observe que não existe nenhuma novidade nesse código em termos de função, temos somente elementos de programação já conhecidos do aluno.

Nas linhas 23 e 24 temos as variáveis *btnMenos* e *btnMais* recebendo os dados de seus respectivos botões. Essas variáveis são testadas nos blocos da função *if*. No bloco da linha 26, o programa verifica se *btnMais* é igual a 1. Perceba que apenas escrevemos o nome da variável e omitimos o sinal de igualdade e o valor de comparação, portanto, podemos dizer que *if(btnMais)* é o mesmo que *if(btnMais == 1)*, da mesma forma como poderíamos dizer que *if(!btnMais)*, note a exclamação, é o

mesmo que *if(btnMais == **0**)*. A exclamação representa uma negação, usada quando queremos fazer uma comparação com o **zero**.

Toda vez que o botão do pino 8 for pressionado, *btnMais* receberá 1 fazendo a variável *pwm* incrementar o seu valor em +5, aumentando a velocidade, isso pode ser visto na linha 28. Na linha 29, temos um if que limita o valor máximo de pwm em **255**, veja que se o valor for incrementado para além desse limite a variável permanecerá em **255**. A função delay nas linhas 30 e 36, servem para que o controle ocorra de forma suave a cada 100 milissegundos, sendo que se diminuir esse tempo, o incremento e o decremento de *pwm* ocorreram mais bruscamente.

Quando o botão do pino 7 for pressionado, a variável *pwm* será decrementada em -5 de seu valor atual, diminuindo a velocidade, conforme mostra a linha 34. Aqui também temos um *if* para limitar o valor, sendo que agora é feito um limite para o valor mínimo, o qual deverá ser igual a zero. Toda vez que *pwm* for decrementado e chegar a receber um valor menor que zero, ela volta a receber zero, permanecendo com esse número. Isso pode ser observado na linha 35.

Por fim, temos a função ***analogWrite**(MOTOR, pwm)*, na linha 39, enviando ao pino 9 os valores obtidos pela variável *pwm* e com isso controlando a velocidade do motor.

Funcionamento

Após a gravação, coloque as pilhas no suporte, vai notar que o motor continuará parado. Pressione e segure o botão S2, com isso a velocidade do motor deverá ir aumentando gradativamente até atingir um valor máximo e assim permanecer. Pressionando o botão S1 e segurando-o, a velocidade deverá começar a diminuir até chegar ao limite zero, onde o motor deverá parar e assim ficar. Note que poderá deixar o motor em qualquer velocidade.

Faça a experiência de trocar o número 5 das linhas 28 e 34, por outros valores que estejam entre 0 e 255, limites do PWM, e veja como o incremento e o decremento irão se comportar. Lembre-se sempre de enviar novamente o código para o Arduino após cada alteração.

MEDINDO DISTÂNCIA COM SENSOR ULTRASSÔNICO

Objetivo

Com esse projeto, o aluno aprenderá usar um dos sensores mais utilizados para realizar a medida de distâncias, servindo também para detectar a aproximação de um objeto como também para aferir o nível em reservatórios. Para esse projeto usaremos cálculos relacionados à velocidade do som, uma

vez que esse sensor é do tipo ultrassônico que para realizar a detecção, ele envia um pulso sonoro que vai até o objeto detectado, reflete e volta para o sensor, técnica usada pelos morcegos existentes na natureza.

Montagem do circuito

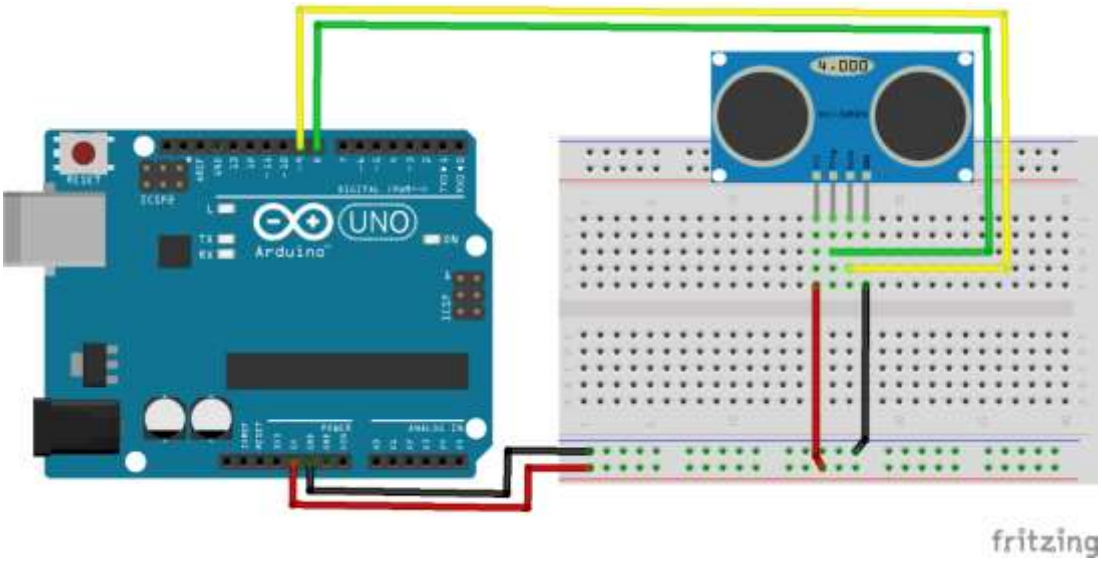


Figura 41 *- Montagem do circuito medidor de distância ultrassônico*

Figura 42 *- Sensor HC-SR04.*

Lista de materiais

Arduino UNO
1 Sensor Ultrassônico HC-SR04
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

A seguir, o código de programação para medida de distância com o Arduino.

```
1  /*****************************************
2   * ------ Medidor de distâncias ------ *
3   *****************************************/
4
5  #define trig 8
6  #define echo 9
7
8  float distancia;
9  long tempo;
10
11 void setup() {
12
13   Serial.begin(9600);
14
15   pinMode(trig, OUTPUT);
16   pinMode(echo, INPUT);
17
18 }
19
20 void loop() {
21
22   digitalWrite(trig, HIGH);
23   delayMicroseconds(10);
24   digitalWrite(trig, LOW);
25
26   tempo = pulseIn(echo, HIGH);
27
28   distancia = (0.0344/2) * tempo;
29
30
31   Serial.println(distancia);
32   delayMicroseconds(5);
33
34 }
```

Explicando o sketch

Boa parte do código já é de conhecimento do aluno. Observe, logo no início, que definimos um pino para o terminal Trigger e outro para o Echo, terminais pertencentes ao sensor. Como a ideia desse projeto é realizar a medida de distâncias, fez-se necessária a declaração de variável do tipo *float*, capaz de armazenar valores com casas decimais. E para a medida de tempo foi declarada uma variável do tipo *long*, este tipo consegue armazenar valores inteiros de até 32 bits (4 bytes), ideal para guardar medidas de tempo.

Partindo para a função *loop()* podemos analisar o funcionamento da seguinte maneira: Na linha 22 temos o *trig* (pino 8 no Arduino) enviando um sinal em nível alto para o sensor, note que esse pulso terá a duração de 10 microssegundos, graças à função presente na linha 23. Essa função,

delayMicroseconds, é nativa da Arduino IDE, assim como a ***delay***, e é usada para se atingir tempos de retardos da ordem de microssegundos. Após esse tempo o *trig* vai à nível baixo, linha 24.

Na linha 26 temos uma outra novidade, a variável *tempo* está recebendo o tempo calculado pela função *pulseIn()*. Essa função entrega para a variável tempo a duração do pulso ***HIGH*** recebido no pino *echo* (pino 9 no Arduino) passado como primeiro parâmetro dessa função. Com isso já temos o tempo de duração do pulso necessário para calcular a distância percorrida pelo sinal do Trigger.

Para se calcular a distância percorrida pelo som iremos precisar dos seguintes dados:

Velocidade do som no ar: 344 m/s

Como nossa medida será em centímetros e o tempo estará em microssegundos, logo devemos converter esse valor em cm/us (centímetro por microssegundos).

344 m x 100 = 34400 cm

1s x 1.000.000 = 1.000.000 us

Logo:

34400/1.000.000 = 0,0344 cm/us

Considerando que o pulso vai até o objeto distante, reflete e volta, portanto, devemos dividir essa velocidade por 2 e com isso o cálculo da distância fica sendo o que temos na linha 28, uma vez que:

$$Distância = \frac{velocidade}{tempo}$$

Como o som será refletido, temos:

$$Distância = \frac{\frac{velocidade}{2}}{tempo}$$

Na linha 31 temos o valor da variável ***distancia*** sendo apresentado no Monitor Serial e na linha 32 temos mais uma pausa no programa de 5 microssegundos para poder ser enviado outro pulso e com isso atualizar a medida.

Funcionamento

Ao conectar o Arduino no computador já com o código gravado, abra o Monitor Serial. Deverá observar de imediato os valores de distância capturados pelo sensor. Posicione o sensor na frente de

algum objeto distante e vá distanciando ou aproximando o sensor do objeto. Com a ajuda de uma régua ou trena na frente do sensor, compare a medida apresentada no Monitor Serial com a medida feita entre o sensor e o objeto.

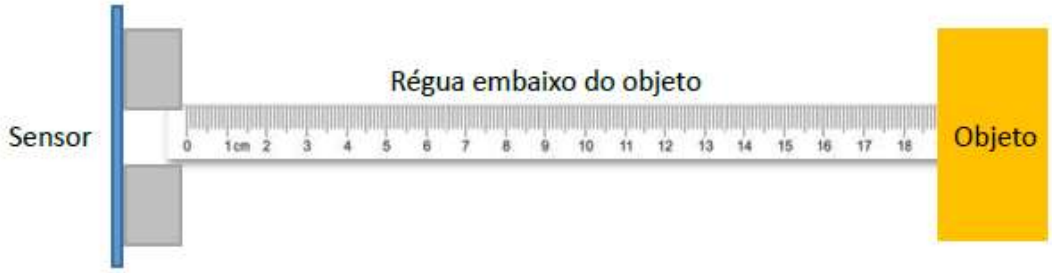


Figura 43 - *Uso de uma régua para comparação da distância medida.*

MEDIDA DE NÍVEL COM SENSOR ULTRASSÔNICO E LCD

Objetivo

A partir desse projeto o aluno aprenderá a fazer uso de bibliotecas para poder controlar alguns componentes e/ou módulos eletrônicos que possam ser usados de forma bastante simples, economizado em tempo de elaboração e digitação de linhas de código. A proposta aqui é semelhantena do projeto anterior, com a diferença de que ao invés de usarmos o Monitor Serial para apresentar os dados na tela do computador, faremos com que a informação seja mostrada em uma pequena tela de cristal líquido.

Montagem do circuito

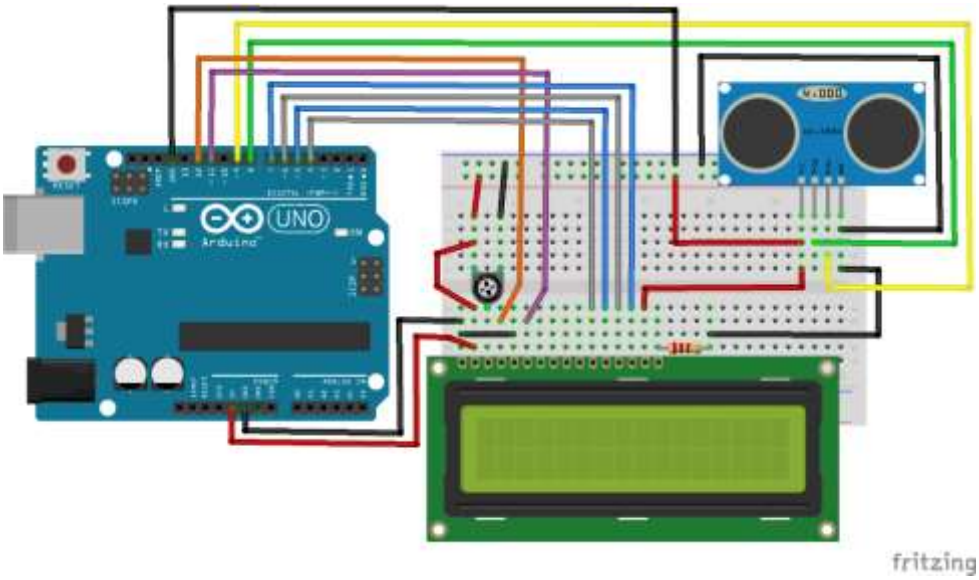


Figura 44 - *Ligação do sensor e do display LCD.*

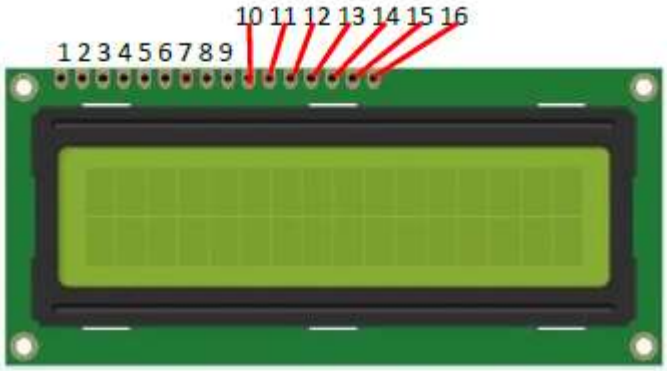


Figura 45 - *Pinout do display LCD 16x2.*

Pinos do LCD

1 – Vss (GND);

2 – Vdd (+5V);

3 – VE ou V0 (Contraste);

4 – *RS (Register Select);*

5 – *RW (Read/Write);*

6 – E ou EN (Enable);

7 ao 14 – *D0 ao D7 (Dados);*

15 – Anodo;

16 – Catodo.

Lista de materiais

Arduino UNO
1 Sensor Ultrassônico HC-SR04
1 Display LCD 16x2
1 Resistor 220Ω (vermelho, vermelho, marrom)
1 Trimpot ou potenciômetro de 5kΩ ou 10kΩ
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

```
/*****************************************
 * ------ Medidor de nível de água ------ *
 *****************************************/

#define trig 8
#define echo 9
#define D4   4
#define D5   5
#define D6   6
#define D7   7
#define EN   11
#define RS   12

#include<LiquidCrystal.h>

float nivel;
long tempo;

LiquidCrystal lcd(RS, EN, D4, D5, D6, D7);

void setup() {

  lcd.begin(16,2);

  pinMode(trig, OUTPUT);
  pinMode(echo, INPUT);
}

void loop() {

  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);

  tempo = pulseIn(echo, HIGH);

  nivel = (0.0342/2) * tempo;

  lcd.setCursor(0,0);
  lcd.print("Nivel medido");
  lcd.setCursor(1,1);
  lcd.print(nivel);
  lcd.print(" cm");

  delay(500);

  lcd.clear();
}
```

Explicando o sketch

A forma como esse código irá usar o sensor ultrassônico para realizar a medida de distância de um objeto foi explicada no projeto anterior e para a medida de nível não será diferente do que já foi apresentado, portanto, iremos dar ênfase ao uso do display LCD com sua biblioteca, buscando compreender cada um dos elementos de programação usados nesse sketch.

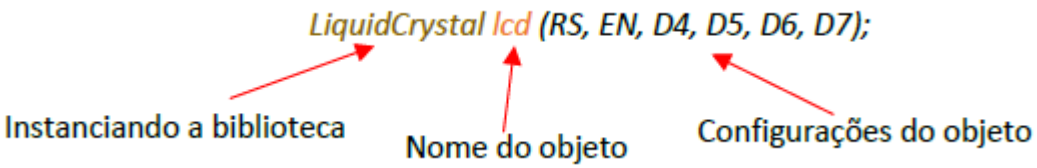
Antes de entrarmos na explicação do código em si, é importante o aluno saber que uma biblioteca é um conjunto de códigos prontos, desenvolvidos geralmente por fabricantes de componentes e módulos eletrônicos ou por programadores experientes que buscam facilitar o processo de

configuração e utilização de um determinado item que deverá interagir com um microcontrolador. As bibliotecas podem ser nativas da Arduino IDE ou podem ser adquiridas e baixadas

em sites de terceiros. Uma dica importante é que, dependendo do que esteja desenvolvendo, busque sempre bibliotecas de fontes confiáveis como as da própria Arduino IDE ou de sites de fabricantes dos componentes. Nunca se sabe o que um programador malicioso acrescentou nas linhas de código.

A biblioteca que está presente nesse sketch é nativa na Arduino IDE, por isso não veremos o processo de buscar e baixar. Observe na linha 14 a diretiva *#include*, a qual está chamando e incluindo no projeto a biblioteca *LiquidCrystal.h*. Os sinais de *"menor que"* e *"maior que"* indica que essa é uma biblioteca armazenada na pasta raiz do compilador do código.

Depois de incluirmos a biblioteca ao nosso projeto, devemos criar o objeto que agirá como um display LCD. Note na linha 19 que foi escrito novamente o nome da biblioteca, mas, agora sem os sinais < e > isso se chama instanciar, e logo em seguida temos escrito ***lcd***. Vejamos:



Após realizarmos a instância da biblioteca devemos dar um nome para o objeto que será criado, que em nosso caso o nome escolhido foi ***lcd*****,** mas poderia ter sido qualquer outro nome, dependerá da imaginação do desenvolvedor, a sugestão é para que sempre nomeie com palavra que tenha coerência com o objeto criado para evitar futuras confusões no decorrer do desenvolvimento. Logo em seguida, para esse tipo de objeto, é repassado os pinos do Arduino que deverão comandar o display e que foram definidos no início do código a partir da linha 7. Devem ser colocados os pinos entre os parênteses na mesma ordem que está sendo mostrada nesse sketch.

Para o Arduino controlar o display LCD, nesse projeto, foram usados apenas dois pinos de comandos o RS e o EN e os quatro últimos de dados que vai de D4 a D7.

Feita a instância da biblioteca e tendo criado o objeto com todas as suas configurações de pinos devemos agora, dentro da função *setup()*, inicializa o objeto *lcd*. Na linha 23, temos a função ***lcd.begin****(16, 2)*, com isso inicializamos o objeto passando como parâmetros a quantidade de colunas e linhas que o display LCD possui. Sendo assim, notamos que o display para esse código deverá ser do tipo 16x2, ou seja, deverá ter 16 colunas e 2 linhas.

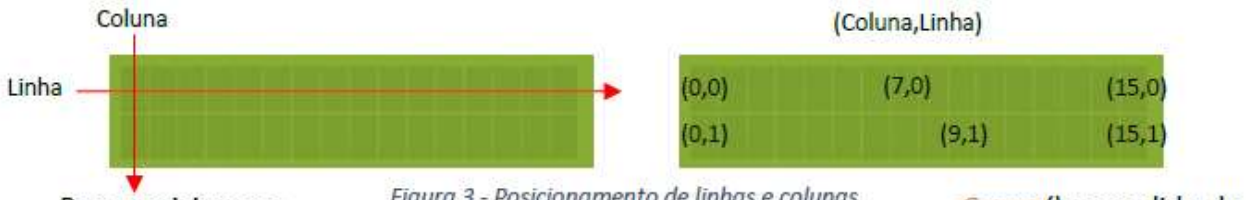


Figura 3 - Posicionamento de linhas e colunas.

Para posicionar o nome do objeto: Cursor() precedido do

lcd.setCursor*(coluna, linha);*

Na linha 39 temos ***lcd.setCursor****(0,0)* isso fará com que a frase *"**Nivel medido**"* inicie na primeira coluna e na primeira linha.

Para escrever uma frase, apresentar um valor na tela ou o resultado de uma variável é usada a função *print()* precedida pelo nome do objeto:

lcd.print("Nivel medido");

Observe que não foi usado acentuação na frase, pois esse tipo de caractere causa erros no display, mas podemos usar sinais gráficos como ponto, interrogação, exclamação, porcentagem entre outros. Veja que na linha 43 foi acrescentado um espaço antes da unidade de medida para que a mesma não ficasse muito junta dos números que irão representar o nível.

Para que os valores sejam mostrados com calma no display, note a presença de um delay(500) na linha 45, isso faz com que os dados sejam atualizados a cada 500 milissegundos.

Por último, na linha 47 temos a função ***lcd.clear****()*, responsável por limpar a tela do display antes de chegar uma nova atualização de valores. Essa função apaga tudo o que tem na tela, mas, como a atualização dos dados está dentro da função *loop()*, então tudo é escrito novamente.

Observação: Caso tivéssemos dado outro nome para o objeto ao invés de ***lcd*****,** como por exemplo *tela*, então todas as funções pertencentes ao display deveriam vir precedidas por essa palavra. Exemplo:

tela.begin*(16,2);*
tela.setCursor*(0,1);*
tela.clear();

FUNCIONAMENTO

Após ter realizada a gravação do código no Arduino, imediatamente o display deverá mostrar na tela a frase Nivel medido, estando essa na primeira linha e iniciando na primeira coluna e logo abaixo deverá ser mostrado os valores capturados pelo sensor ultrassônico. Esses valores deverão aparecer

a partir da segunda coluna e na segunda linha e deve haver um espaço entre os números e a unidade de medida.

Ajuste o trimpot P1 para obter o melhor contraste na tela. Caso não esteja apresentando informações incompreensíveis, reveja com calma e/ou refaça as ligações do display.

DECIBELÍMETRO COM LCD

Objetivo

O principal objetivo desse projeto, além de praticar o que já foi apresentado em projetos anteriores, é mostrar como usar componentes eletrônicos discretos para a elaboração de um circuito, que, nesse caso, terá a capacidade de captar sinais de áudio por meio de um microfone de eletreto acoplado a um pequeno e simples amplificador de áudio transistorizado. A princípio vai parecer complicado realizar essa montagem, mas é só compreender o significado de cada um dos símbolos que serão apresentados junto com a montagem do circuito que tudo ficará claro.

Montagem do circuito

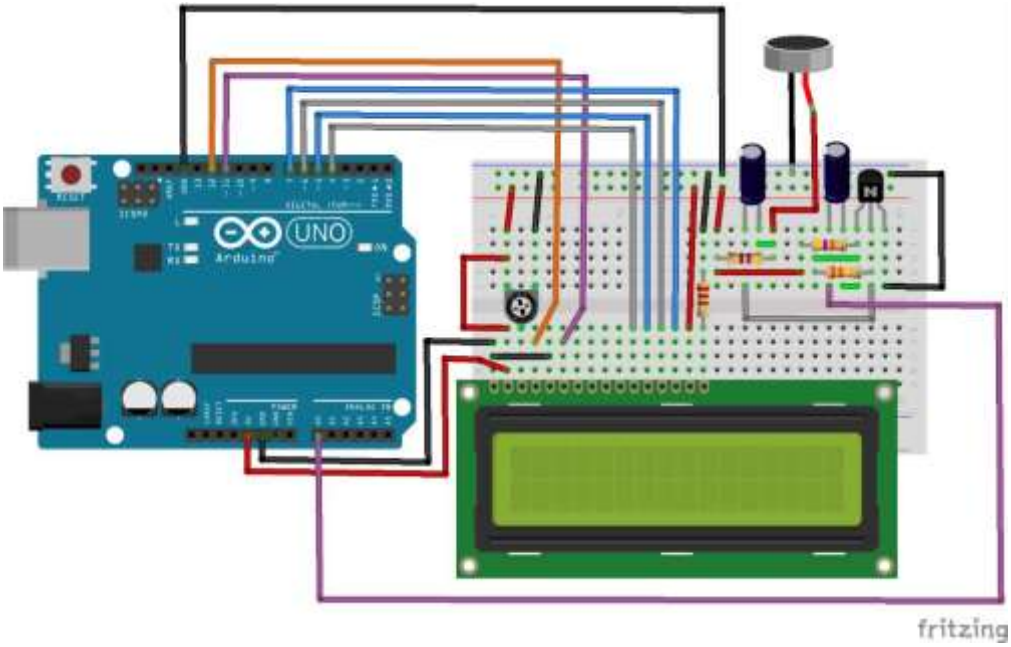


***Figura 46** - Montagem do circuito.*

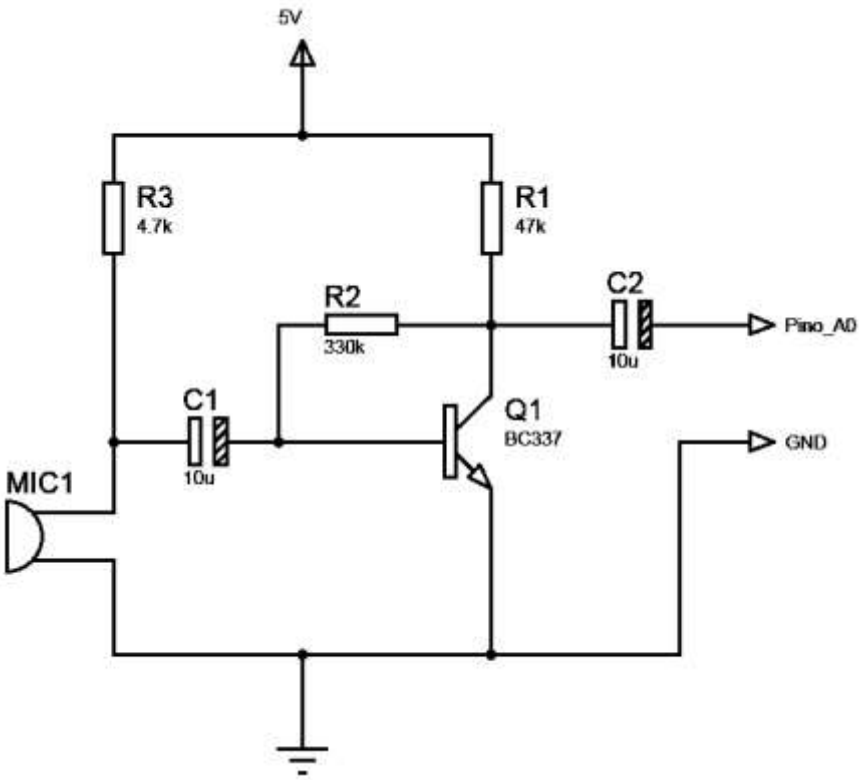


Figura 47 – *Esquema elétrico do circuito eletrônico do amplificador para microfone de eletreto.*

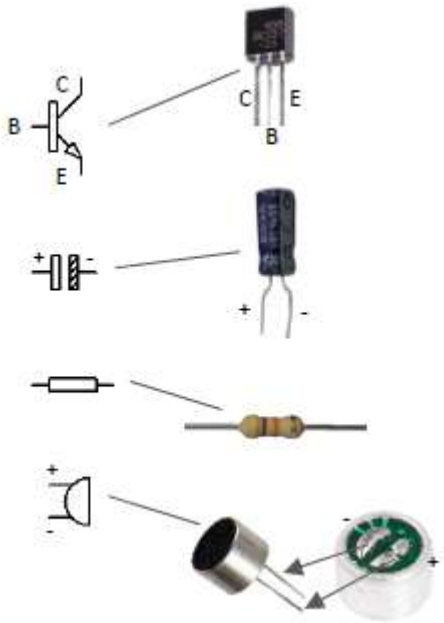


Figura 48 – *Simbologia e polaridade dos terminais.*

Na realização da montagem, principalmente do circuito do microfone, é importante observar as corretas polaridades dos terminais de cada um dos componentes, simbolizadas pelos sinais de positivo (+) e negativo (-), há uma ressalva apenas para os resistores, os quais, não possuem polaridade definida. O transistor possui seus terminais nomeados como Coletor (C), Base (B) e Emissor (E).

Observe bem as figuras e certifique-se de que todos os componentes estejam interligados da maneira mostrada nas figuras 1 e 2. A figura 2 é a representação esquemática do circuito do microfone.

Lista de materiais

Arduino UNO
1 Microfone de eletreto
1 Display LCD 16x2
1 Resistor 220Ω (vermelho, vermelho, marrom) para R4
1 Resistor 4,7kΩ (amarelo, violeta, vermelho) para R3
1 Resistor 47kΩ (amarelo, violeta, laranja) para R1
1 Resistor 330kΩ (laranja, laranja, amarelo) para R2
1 Transistor BC337 ou BC549 para Q1
2 Capacitores eletrolíticos 10uF x 16V para C1 e C2
1 Trimpot ou potenciômetro de 5kΩ ou 10kΩ para P1
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

Digite o código abaixo na Arduino IDE e envie para a placa.

```
/*****************************************
 * ------ Medidor de decibel com LCD------ *
 *****************************************/

//Definições do LCD
#define D4 4
#define D5 5
#define D6 6
#define D7 7
#define EN 11
#define RS 12
#define som analogRead(A0)

#include <LiquidCrystal.h>

LiquidCrystal tela(RS, EN, D4, D5, D6, D7);

float dB,volts;
int adc;

void setup() {

  tela.begin(16,2);

  tela.setCursor(0,0);
  tela.print("Decibelimetro");
  delay(1500);
  tela.clear();
}

void loop() {

  adc = som;
  volts = adc*5.0/1023;

  dB = 15.15 + 1.56*(volts*volts*volts) - 12.34*(volts*volts) + 30.39*volts;

  tela.setCursor(0,0);
  tela.print("Medidor decibel");
  tela.setCursor(0,1);
  tela.print(dB, DEC);
  tela.print(" dB");

  delay(500);

  tela.clear();
}
```

EXPLICANDO O SKETCH

Esse sketch é bastante semelhante com o apresentado no projeto anterior, mas logo no início, entre as diretivas de definição, podemos ver algo novo que não havia sido apresentado ainda. Na linha 12 temos uma definição chamada *som* e foi atribuída não a um pino, mas, sim a uma função, que nesse caso foi a função ***analogRead****(A0)*, assim, toda vez que precisarmos fazer a leitura do pino analógico *A0*, nos bastará apenas escrever a palavra *som*. Esse recurso é muito útil quando se tem um projeto grande, onde é necessário chamar diversas vezes uma função para controle ou leitura de pinos.

As explicações sobre a biblioteca e configuração para uso do display LCD já foram apresentadas no último projeto, portanto, não iremos tecer maiores comentários, mas note que nesse sketch, o objeto de ***LiquidCrystal*** foi nomeado como *tela*, isso foi feito para que o aluno perceba que poderá sempre nomear um objeto da maneira que achar mais apropriado.

Para este projeto temos duas variáveis do tipo *float*, nomeadas como *dB* e *volts*, e uma outra do tipo *int* chamada de *adc*. Suas declarações são mostradas nas linhas 18 e 19, respectivamente.

Na função *setup()*, a partir da linha 25 até a 28 faremos com que a palavra "*Decibelimetro*" apareça na tela por 1,5 segundos (1500 milissegundos), desaparecendo logo em seguida graças a função ***tela.clear****()*.

Partimos agora para a função *loop()*. Sabemos que a porta analógica entrega para nosso programa, valores que podem variar de zero a 1023 (porta *A0* em nosso caso) e para que possamos converter esses níveis de sinais em decibel, primeiramente iremos convertê-los em tensão elétrica. A variável *adc* recebe os sinais analógicos vindos da porta *A0* por meio da definição *som*, observe isso na linha 33. Para que esses valores analógicos sejam convertidos em valores de tensão elétrica, na linha 34 temos a variável *volts* recebendo o resultado da seguinte fórmula:

$$volts = adc * \mathbf{5.0/1023};$$

Essa fórmula foi apresentada e estudada no projeto "*VOLTÍMETRO NO MONITOR SERIAL*", caso o aluno esteja com dúvida será importante voltar e dar uma olhada nesse projeto.

Esse valor de tensão armazenado na variável *volts* é proporcional à intensidade sonora capturada pelo circuito do microfone e para que possamos converter em valores de decibel precisamos usar a fórmula da linha 36:

$$dB = \mathbf{15.15} + \mathbf{1.56} * (volts * volts * volts) - \mathbf{12.34} * (volts * volts) + \mathbf{38.39} * volts;$$

Essa fórmula vem de:

$$dB=15{,}15+1{,}56volts^3-12{,}34volts^2+38{,}39volts$$

A qual corresponde ao cálculo simplificado do decibel usando a função de interpolação polinomial de terceiro grau. A fórmula completa para o cálculo do decibel a partir de sinais de tensão elétrica é:

$$dBv = 20log\left(\frac{v1}{v0}\right)$$

Não entraremos em detalhes a respeito dessa fórmula, apresentamos apenas para título de esclarecimento do que foi usado no sketch.

Após o cálculo, os valores em decibel serão mostrados na tela do display por meio das funções escritas a partir da linha 38 até a linha 42. Observe na linha 41 a presença da sigla *DEC*. Essa sigla serve para converter, em tempo de execução da função *tela.print(),* um valor para decimal mostrando o resultado nesse formato. O resultado permanecerá sendo mostrado na tela por 500 milissegundos, como podemos ver na linha 44, sendo posteriormente apagado por completo graças a função ***tela.clear()*** da linha 46. Após isso, tudo dentro de *loop()* volta a se repetir atualizando os valores.

Funcionamento

Após gravar o código na placa do Arduino lembre-se de ajustar o *trimpot* P1 para obter o melhor contraste na tela. Caso não esteja apresentando informações compreensíveis, reveja com calma e/ou refaça as ligações do display.

Estando tudo certo com o display, toda vez que o Arduino for ligado ou ter seu botão de *reset* pressionado, a palavra “Decibelimetro” deverá aparecer na tela por 1,5 segundos e depois desaparecer, dando lugar à frase “Medidor decibel” e na linha abaixo o valor decimal correspondente à intensidade sonora deverá aparecer variando conforme o som ambiente. Quanto maior o barulho, maior será o valor apresentado.

CONTROLANDO SERVO-MOTOR

Objetivo

Neste projeto o aluno passará a conhecer um servo-motor e uma das maneiras de realizar o controle de seu eixo. A programação será bem simples e o entendimento mais simples ainda. Usaremos um potenciômetro para fazer o controle do servo-motor, por isso, veremos mais uma vez a conversão de

sinal analógico para digital, bem como a adequação dos valores. Apesar da simplicidade do circuito e do código, este conteúdo irá abrir um leque de ideias para futuros projetos.

Montagem do circuito

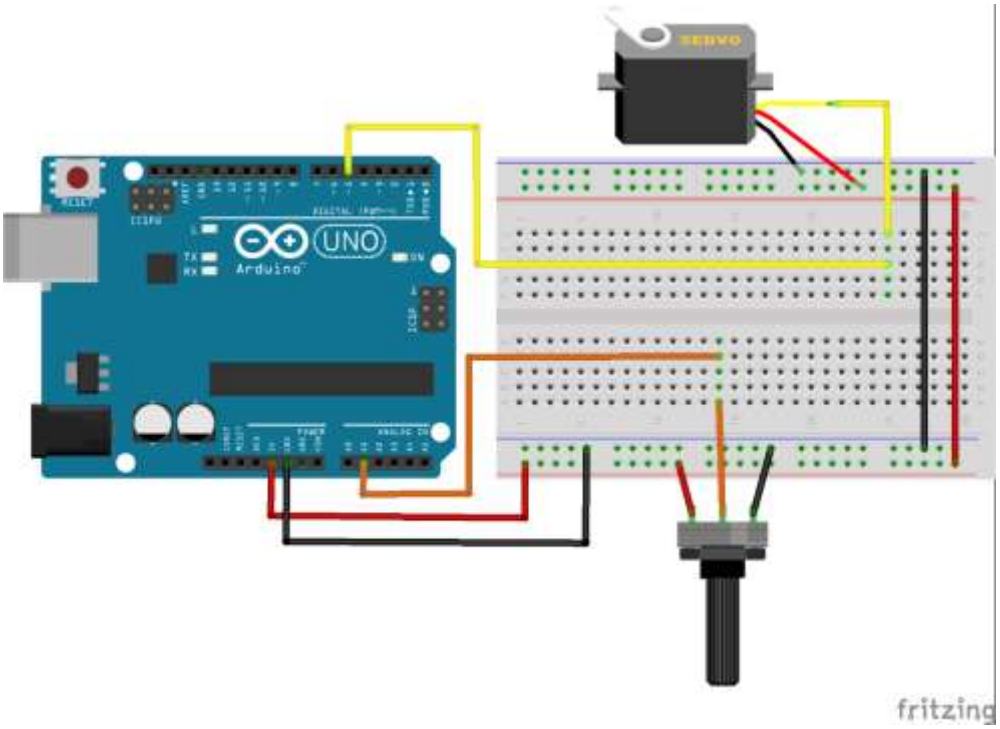


Figura 49 - *Montagem do circuito que controla um servo-motor.*

Figura 50 - *Servo-Motor.*

O servo-motor é um tipo de motor usado para posicionamento. A maioria tem a rotação máxima de seu eixo limitada a 180°, mas existem alguns que se movem até os 360°.

Possui três terminais:

- Vermelho: Terminal positivo;
- Preto ou marrom: Terminal negativo;
- Amarelo, laranja ou branco: Terminal de controle.

A tensão de alimentação usada para o tipo de servo-motor que usaremos aqui é de 5V, no entanto, existem outros com diferentes tipos de tensão nominal, por isso, sempre verifique o manual ou datasheet do modelo que irá empregar em um projeto.

Outra informação importante a se saber no momento da escolha de um servo-motor é a capacidade dele em Força Peso - kgf (quilograma força), dependendo do que ele irá movimentar, esse dado será muito útil para garantir que o trabalho seja executado sem problemas.

Lista de materiais

Arduino UNO
1 Servo-motor SG90 ou similar para M1
1 Potenciômetro 5k ou 10k para P1
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

Digite o código mostrado abaixo em sua Arduino IDE e envie para a placa.

```
/*****************************************
 * ------ Controlando Servo-Motor ------ *
 *****************************************/

#include <Servo.h>

int pot, angulo;

Servo motor;

void setup() {

  motor.attach(5);

}

void loop() {

  pot = analogRead(A1);

  angulo = map(pot, 0, 1023, 0, 180);

  motor.write(angulo);

}
```

EXPLICANDO O SKETCH

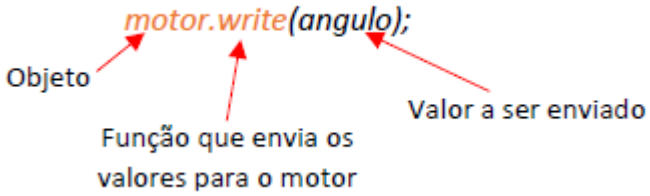
Como pode observar, o código é bastante simples, graças ao uso da biblioteca ***Servo.h*** que incluímos na linha 5. Ela é responsável por realizar os cálculos e gerar os pulsos necessários para que o servo-motor possa mover seu eixo em um ângulo específico. Sem essa biblioteca, esse código seria bem maior e mais complexo, pois teríamos que realizar os cálculos que resultariam em movimento angular a partir de um sinal de onda quadrada gerada a partir de uma precisa base de tempo.

Na linha 7 temos as declarações das duas únicas variáveis do tipo ***int*** que iremos precisar, sendo a de nome *pot* para armazenar os valores provenientes do pino analógico *A1* e a variável *angulo* para guardar os valores convertidos em ângulos e assim ser passada para a função que fará o movimento do eixo.

Na linha 9 nos deparamos com a criação do objeto Servo, o qual damos o nome de ***motor***, veja que não foi necessário passar nenhuma informação de configuração para esse objeto, como foi feito no projeto anterior para a criação do objeto ***lcd***. Já dentro do *setup()*, na linha 13, temos a função *motor.attach(5)*, responsável por configurar o pino 5 do Arduino como saída de pulso para controle do servo-motor. Pode ser escolhido qualquer outro pino digital, substituindo o número *5* pelo número do pino desejado, e depois é só conectar o terminal de controle do servo-motor a ele.

Na função *loop()*, a variável *pot*, linha 19, recebe os valores com intervalos de **zero** a **1023**, provenientes do canal analógico *A1*, o qual depende da posição do cursor do potenciômetro. No entanto, como sabemos, nosso motor só consegue atingir um ângulo máximo de 180° na rotação de seu eixo, ou seja, os valores para realizar o controle devem pertencer ao intervalo que vai de zero a 180, no entanto, o potenciômetro estará entregando um range bem maior que esse. Para contornar o problema exposto, na linha 21, a variável *angulo* receberá o intervalo de valores ajustados pela função ***map****(pot,* ***0, 1023, 0, 180****)*, essa função já foi estudada no projeto "CONTROLANDO BRILHO DE LED POR POTENCIÔMETRO" por isso não entraremos em maiores detalhes.

Com a adequação dos valores concluída pela função *map*, a variável *angulo* receberá valores dentro da faixa pretendida que é de **zero** a **180** o que possibilitará mover o eixo do servo-motor para posições que irão até 180°. O valor dessa variável é passada para o servo-motor por meio da seguinte função:



Olhando para a função acima, note que podemos colocar qualquer valor de zero a 180 no lugar da variável *angulo* para que o servo-motor permaneça fixo em determinada posição.

Projetos como este são bastante úteis na indústria, imagine conseguir mover uma válvula pesada girando apenas o cursor de um potenciômetro, ou mover câmeras de vigilância para verificar um determinado perímetro. São diversas as possibilidades, tudo dependerá da necessidade e de sua criatividade.

Funcionamento

Para realizar o teste desse circuito é bem simples. Ao carregar o código na placa Arduino, o servo-motor poderá mover seu eixo para uma posição que dependerá da posição do cursor do potenciômetro no momento. Girando o cursor do potenciômetro, o eixo do servo-motor deverá acompanhar o movimento. Deixando o cursor em uma certa posição o servo-motor deverá se manter na posição em que parou e apenas a aplicação de uma força maior que a sua capacidade poderá tirá-lo dessa posição enquanto o circuito estiver ligado, mas, não é recomendável sobrecarregar o eixo com força peso maior que sua capacidade, isso poderá danificar as engrenagens internas.

CONTROLANDO SERVO-MOTOR COM SENSOR ULTRASSÔNICO

Objetivo

Neste projeto iremos praticar o que foi apresentado nos assuntos anteriores fazendo um sensor ultrassônico comandar um servo-motor, dois elementos já conhecidos do aluno. Circuitos como esse são bastante úteis quando se precisa acionar um atuador quando um objeto ou indivíduo se aproxima a determinada distância, por exemplo, a abertura de portas com a aproximação de alguém, o levantamento de cancelas para passagens de veículos, o acionamento de esteiras na presença de um objeto, entre muitas outras aplicações.

Montagem do circuito

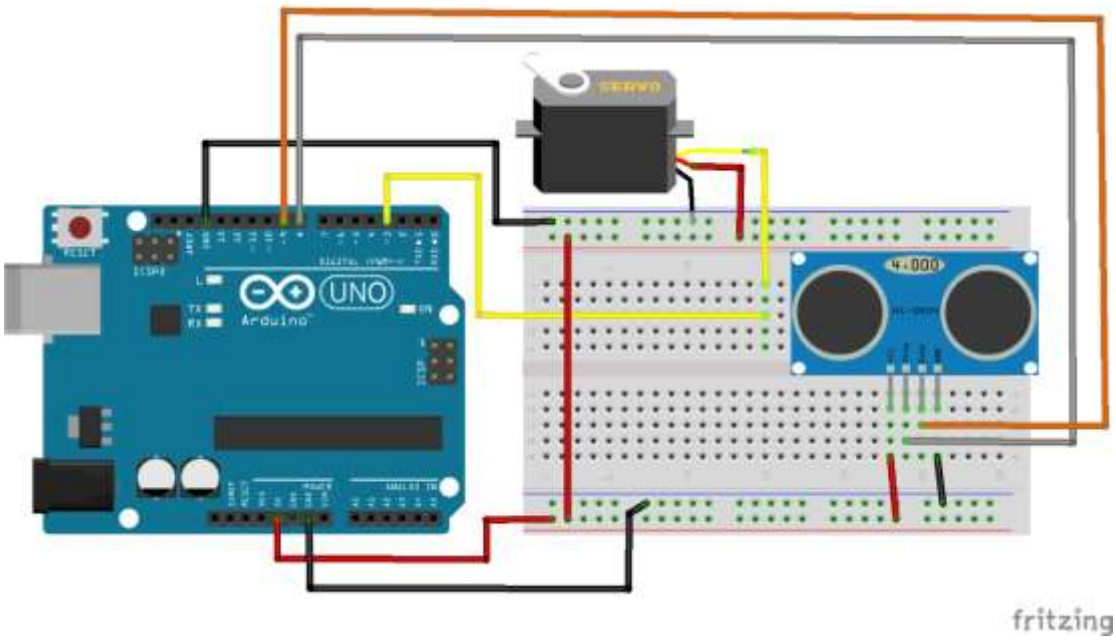


Figura 51 - *Montagem do sensor e do servo-motor.*

A montagem é bem simples, como se pode ver, e os detalhes sobre os componentes usados já foram apresentados em projetos anteriores. Sempre tome o cuidado de estar atento na correta ligação dos cabos para que tudo funcione sem problemas.

Lista de materiais

Arduino UNO
1 Sensor Ultrassônico HC-SR04
1 Servo-motor SG90 ou similar
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

Após digitar o código mostrado abaixo, carregue para a placa Arduino.

```
1  /*************************************************
2   * ------ Servo-Motor e sensor ultrassônico ------ *
3   *************************************************/
4
5  //Definições dos pinos para o sensor ultrassônico.
6  #define trig 8
7  #define echo 9
8
9  #include <Servo.h>
10
11 int dist;
12 long tempo;
13
14 Servo motor;
15
16 void setup() {
17
18   motor.attach(3);
19
20   pinMode(trig, OUTPUT);
21   pinMode(echo, INPUT);
22 }
23
24 void loop() {
25
26   digitalWrite(trig, HIGH);
27   delayMicroseconds(10);
28   digitalWrite(trig, LOW);
29
30   tempo = pulseIn(echo, HIGH);
31
32   dist = (0.0342/2) * tempo;
33
34   if(dist < 15)
35   {
36     motor.write(90);
37   }
38   else
39   {
40     motor.write(0);
41   }
42
43
44 }
```

EXPLICANDO O SKETCH

Esse sketch não tem muito o que explicar, uma vez que, se o aluno observar o que temos aqui verá que nada mais é do que a união de dois outros sketchs já estudados, onde foram apresentados os detalhes de uso do sensor ultrassônico e do servo-motor.

Na parte inicial temos as definições dos pinos do sensor nas linhas 6 e 7, na linha 9 vemos a inclusão da biblioteca ***Servo.h*** ao projeto, nas linha 11 e 12 podemos ver as declarações das variáveis usadas no projeto e na linha 14 encontramos a criação do objeto ***motor*.** Dentro da função *setup()*, estão as configurações dos pinos, tanto do que irá controlar o servo-motor, quanto os de emissão (Trigger) e recepção (echo) de pulsos do sensor ultrassônico. E na função *loop()*, da linha 26 até a 32, temos os elementos de programação responsáveis pela geração de pulsos e cálculos da distância medida pelo

sensor ultrassônico, os detalhes sobre esses elementos foram estudados no projeto "MEDINDO DISTÂNCIA COM SENSOR ULTRASSÔNICO", portanto, não teceremos maiores explicações.

O segredo desse projeto está no módulo *if* mostrado a partir da linha 34, note que aqui a variável *dist* está sendo comparada com o valor **15,** ou seja, verifica se o valor que ela recebeu proveniente do sensor é menor que **15 cm**, caso essa sentença seja verdadeira o comando da linha 36 será executado, fazendo o eixo do servo-motor girar até a posição de **90°.** Mas, caso a sentença seja falsa, ou seja, se o valor da variável *dist* não for menor que 15 cm, então o comando que será executado será o da linha 40, o que fará o servo-motor mover seu eixo para a posição de **0°**.

Funcionamento

Após ter carregado o código para a placa Arduino, basta aproximar um objeto ou mesmo a mão na frente do sensor ultrassônico, quando a distância entre o objeto e o sensor for menor que **15 cm**, o eixo do servo-motor deverá mover-se para a posição de **90°** e quando o objeto for afastado a uma distância maior ou igual a **15 cm**, o eixo deverá voltar para a posição de **0°.** Perceba que poderá testar outros valores de ângulos para um movimento mais aberto ou mais fechado, lembrando apenas que a faixa de valores que podem ser usados para realizar o controle do servo-motor varia de **0** a **180**.

Quando aproximar ou afastar o objeto do sensor bem devagar, notará que o servo-motor irá vibrar como se não soubesse o que fazer e isso sempre ocorrerá quando o objeto estiver entre **14** cm e **15 cm** de distância. Como forma de exercitar a capacidade de resolver problemas, tente encontrar uma forma de resolver essa questão e fazer com que o movimento do eixo se torne mais suave.

CONTROLE REMOTO POR RF

Objetivo

Neste projeto o aluno irá aprender a trabalhar com transmissão de dados via Rádio Frequência (RF) usando os módulos RXTX na construção de um controle remoto simples. O conhecimento adquirido será de grande importância para o desenvolvimento de equipamentos que necessitem ser controlados por sistema de rádio, o que é muito útil quando necessitamos acionar algo à distância em ambiente que não tem acesso à rede WiFi e onde um sistema via Bluetooth não teria um alcance adequado. Além do exposto, será apresentado alguns elementos de programação referentes à manipulação de caracteres.

Montagem do circuito

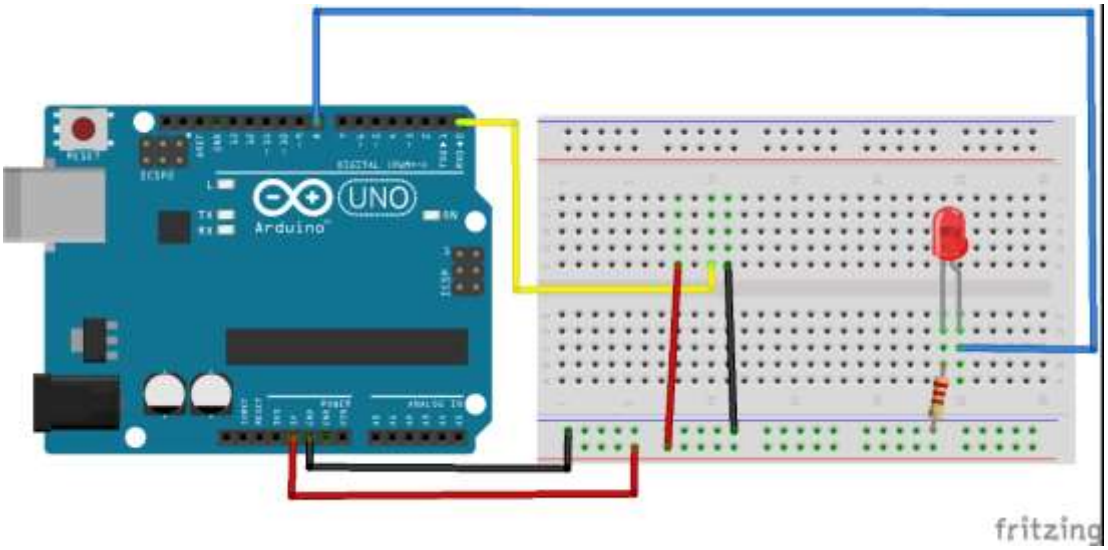


Figura 52 - *Circuito transmissor com módulo TX.*

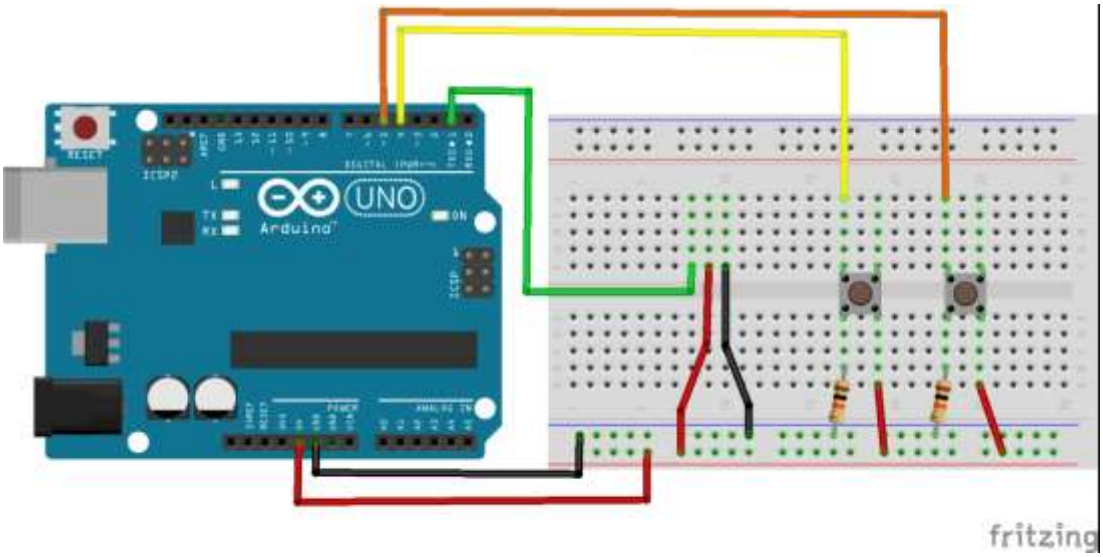


Figura 53 - *Circuito receptor com o módulo RX.*

Os módulos RX TX de Rádio Frequência

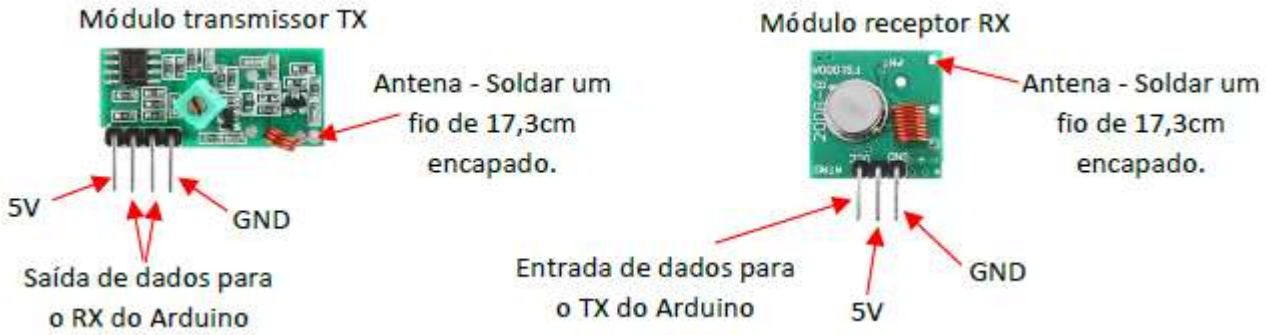


Figura 54 - *Pinout dos módulos transmissor e receptor.*

Esses módulos trabalham com frequência de portadora de 315MHz e também de 433MHz, sendo essa última a mais comum. Podem atingir um alcance de até 100m em área livre com o uso de uma antena, que nada mais é do que um pedaço de fio rígido encapado soldado um em cada módulo no local indicado na figura 3. Para o correto comprimento das antenas, tanto do transmissor quanto do receptor, pode-se usar o cálculo mostrado abaixo e usaremos 433MHz como exemplo:

$$Antena = \frac{\lambda}{4} = \frac{\frac{c}{f}}{4}$$

Onde:

λ=$Comprimento\ de\ onda\ em\ metros$;

c=$Velocidade\ da\ luz\ no\ vácuo\ em\ metros\ por\ segundo$;

f=$Frequência\ de\ operação\ em\ Hz$.

$$Antena = \frac{\frac{3 \times 10^8}{433 \times 10^6}}{4} \cong 0{,}1732m$$

Com isso vemos que as antenas deverão ter aproximadamente 17,32cm de comprimento.

No mercado existem diversos modelos de módulos RX TX, no entanto, os pinos são geralmente os mesmos mostrados aqui, mas sempre verifique com atenção se os módulos que estará prestes a comprar possuem os pinos que necessita para seu projeto. São sempre vendidos em pares e caso os encontre separadamente, verifique se ambos operam na mesma frequência.

Lista de materiais

2 Arduinos UNO
1 Módulo RX 315MHz ou 433MHz
1 Módulo TX 315MHz ou 433MHz
1 LED vermelho, verde ou amarelo
1 Resistor 220Ω (vermelho, vermelho, marrom)
2 Resistor 10kΩ (marrom, preto, laranja)
2 Botões táteis do tipo Push-button
Protoboard
Fios ou cabinhos jumpers
Cabo USB

Programação

Este projeto contará com dois sketchs de programação, um para o transmissor e outro para o receptor. Digite cada um em arquivos separados da Arduino IDE e depois envie cada um para suas respectivas placas.

```
/*****************************************
 * ------ Transmissor de dados RF ------ *
 *****************************************/

#define LIGAR digitalRead(4)
#define DESLIGAR digitalRead(5)

void setup()
{
  Serial.begin(1200);

  pinMode(4, INPUT);
  pinMode(5, INPUT);
}

void loop()
{
  if(LIGAR)
  {
   Serial.write("ligar&");
   Serial.flush();
  }
  else if(DESLIGAR)
  {
    Serial.write("des&");
    Serial.flush();
  }
}
```

```
/*************************************
 * ------ Receptor de dados RF ------ *
 *************************************/

String info;

void setup()
{
  Serial.begin(1200);

  pinMode(8, OUTPUT);
}

void loop()
{
  if (Serial.available())
  {
    char dado = Serial.read();

    if(dado == '&')
    {
      Serial.println(info);

      if(info.indexOf("ligar") > -1)
      {
        digitalWrite(8, HIGH);
      }
      else if(info.indexOf("des") > -1)
      {
        digitalWrite(8, LOW);
      }

      info.remove(0);
    }
    else
    {
      info += dado;
    }
  }
}
```

EXPLICANDO O SKETCH

Começaremos explicando o código do transmissor. Inicialmente não temos nenhuma novidade nas primeiras linhas desse código, note apenas que usamos como velocidade de transmissão de dados para a comunicação serial o valor de ***1200bps***, a qual não deve ser maior que isso para poder manter a integridade dos dados que serão transmitidos. Velocidades maiores fazem com que os dados cheguem ao receptor de formas incompreensíveis.

Nas linhas 12 e 13, configuramos os pinos 4 e 5 como entradas digitais para que o Arduino possa fazer a leitura dos botões S1 e S2, presentes no circuito da figura 2. Os pinos citados foram definidos nas linhas 5 e 6 com o nome ***LIGAR*** para o pino de entrada 4 e ***DESLIGAR*** para o pino de entrada 5.

Dentro da função *loop()*, a partir da linha 18, nos deparamos com os blocos de *if* que irão verificar quais dos dois botões foram pressionados. Se o botão S1, o qual está instalado no pino 4, for pressionado, o programa enviará pela serial o comando ***"ligar&"*** e isso será feito com a função ***Serial.write("ligar&").*** A função *write* tem a capacidade de enviar byte a byte de uma informação via comunicação serial, o que é ideal para sistemas de controle como esse. Logo abaixo, na linha 21, temos a função ***Serial.flush()***, ela não recebe nenhum parâmetro, por isso seus parênteses estão vazios e tem a finalidade de fazer o programa esperar até que toda a informação seja enviada.

Na linha 23, o *else if* verifica se o botão S2 foi pressionado, em caso afirmativo, será enviado o comando *"des&"* e isso ocorrerá usando os mesmos elementos de programação explicado, linhas atrás. Perceba que não usamos a palavra "*desligar"* para ser enviada, isso será entendido melhor na explicação do código do receptor.

Resumindo, o sketch do transmissor ficará esperando um dos botões ser pressionado, caso seja S1, o programa enviará o comando "***ligar&"*** e se for pressionado o botão S2, o comando a ser enviado será "*des&"*. O caractere "*&"* também será entendido com a explicação do próximo código.

No sketch do receptor encontraremos algumas novidades, o que já pode ser notado na linha 5 com a declaração de uma variável do tipo *String*. Variáveis desse tipo conseguem armazenar palavras ou mesmo textos, ou seja, guardam cadeias de caracteres. Aqui declaramos nossa *String* com o nome de *info* e será usada para receber os comandos enviados pelo transmissor.

Na função *setup()* não temos nenhuma novidade, apenas observe que o valor de baud rate também é de ***1200bps***, perceba que tanto no transmissor, quanto no receptor, essa taxa deve ter o mesmo valor, caso contrário os dados transmitidos não serão compreendidos pelo circuito receptor e, como dito

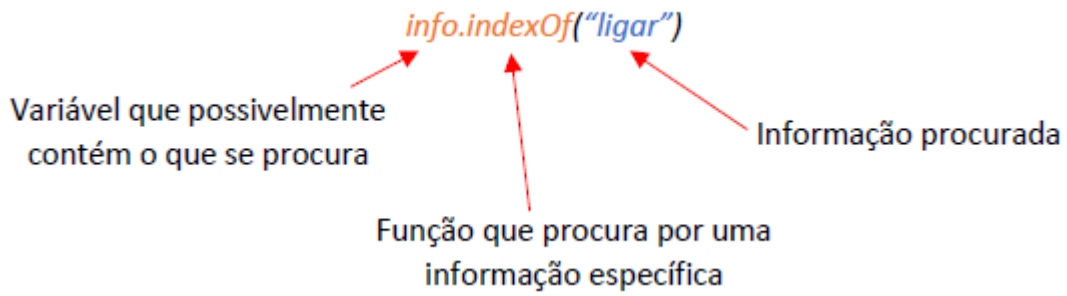
anteriormente, não é recomendado ser um valor maior que o usado nesses sketchs, por motivo já explicado.

O receptor terá a responsabilidade de ligar e desligar um LED toda vez que receber o comando específico para cada tarefa. O LED instalado no pino 8 do Arduino será comandado à distância pelo transmissor e o correto funcionamento se dará graças às funções presentes no *loop()*. Na linha 16, o programa começa usando um recurso bem interessante que é a função ***Serial.available()***, essa função tem a finalidade de ficar verificando se chegou alguma informação pela porta de comunicação Serial, ou seja, se o pino RX, ou a USB do Arduino recebeu alguma informação, se sim, a função retornará um valor verdadeiro, o qual será analisado pelo *if*. Caso tenha recebido alguma informação, então será analisada se ela é para ligar o LED ou desligá-lo, mas se não receber nada o programa permanecerá ocioso.

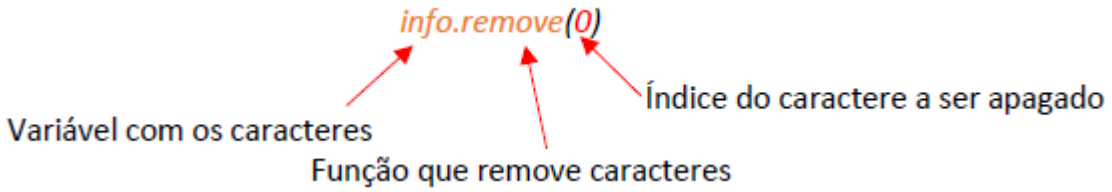
Partindo do ponto de vista que o Arduino recebeu uma informação, vejamos como ocorrerá o processo de análise. A primeira coisa a ser feita é capturar cada byte de informação que está chegando, por isso, na linha 18 temos a declaração de uma variável local do tipo *char* chamada *dado*. Variáveis desse tipo só conseguem armazenar um caractere por vez, ou seja, apenas um byte. Vamos supor que tenha chegado o comando ***"ligar&",*** com isso a variável *dado* armazenará primeiramente o caractere *"l"* (desconsidere as aspas elas não vêm do transmissor). Com esse caractere capturado, o *if* da linha 20 irá verificar se o que chegou é igual ao caractere ***"&"***, como não é, então, o programa passa para o *else* da linha 37, onde temos a variável *info* recebendo o caractere que chegou. Nesse ponto observe o operador de associação +=, esse operador armazena uma informação em uma variável do tipo *String* sem apagar o que já foi guardado, sendo assim, o *"l"* que chegou, ficará na variável *info* e será associada a outro caractere que chegar. O próximo caractere passado para a variável *dado* é o *"i"*, por sua vez, esse também não é igual a ***"&"***, portanto, o programa novamente vai para a linha 37 e coloca esse caractere na variável *info*, com isso, já temos armazenado nessa variável os caracteres "li" graças ao operador de associação. A cada ciclo em busca do caractere ***"&"***, a palavra ***"ligar"*** vai sendo formada dentro da variável *info*. Quando a variável *dado* receber ***"&"***, então a análise feita na linha 20 será verdadeira e, com isso, o programa não irá mais para a linha 37.

Na linha 22 será enviada, para poder ser visualizada no Monitor Serial, a palavra armazenada na variável *info*. Esse comando não é obrigatório para o funcionamento de nosso circuito, mas ajudará o aluno a ver como a palavra está formada. Nas linhas 24 e 28 temos mais uma novidade que é o uso da função *indexOf()*. Essa função, atrelada à variável *info*, permite buscar, entre diversos elementos de

um texto, uma letra, palavra ou frase específica e no caso da linha 24 ela está buscando pela palavra "ligar". Sua sintaxe fica sendo:



Quando o ***indexOf()*** encontra o que procura, ele entrega, implicitamente, para a função *if* um valor maior que -1, esse valor corresponde ao local em índice onde a informação foi encontrada, mas isso é detalhe para outro assunto, por enquanto basta entender que quando essa função encontra o que procura ela entrega um valor maior que -1. Como em nossa suposição a variável *info* realmente contém o que se procura, logo o programa irá executar o comando da linha 26 que corresponde em enviar *HIGH* para o pino 8 do Arduino fazendo o LED acender. Logo depois de executar essa tarefa, o programa vai para a linha 33 e veja, que aqui, temos mais uma novidade, a função *remove(0)* associada à variável *info*. Essa função apagará todos os caracteres existentes em *info*, deixando-a vazia para poder receber novos dados. Para entender um pouco mais sobre essa função, vejamos:



Cada caractere armazenado em uma *String* recebe um valor de índice começando pelo zero, assim, na palavra *"**liga**r"* o caractere ***"l"*** teria o índice *0* (zero), o ***"i"*** o índice *1* e assim sucessivamente. A função ***remove()*** apaga os caracteres a partir de um índice inicial, como colocamos o valor zero entre seus parênteses, então serão apagados todos os caracteres partindo do que começa com o zero até o último. Se colocássemos, por exemplo, o valor *2*, seriam apagados os caracteres ***"gar",*** ficando apenas ***"li"*** na variável *info*. O interessante dessa função é que podemos passar até dois parâmetros, sendo o primeiro o valor do índice a partir do qual queremos começar a eliminar os caracteres e um outro valor maior que corresponde ao valor a partir do, qual queremos deixar. Exemplo:

info.remove(1,3)

Dessa forma, da palavra ***"ligar"*** presente na variável *info* serão removidos os caracteres com índice *1* e *2* que **são *"i"*** e ***"g"*** e, assim, permanecerão na variável os caracteres ***"lar",*** ou seja, os caracteres com índices *0*, ***3*** e ***4***. Note que o índice *3* pertence ao caractere ***"a"*** e como esse foi passado como último parâmetro, então permaneceu.

Depois do exposto fica claro entender o porquê de não termos escolhido a palavra **"desligar&"** como comando para ser enviado pelo transmissor. Se essa palavra fosse usada, perceba que nela também existe a palavra **"ligar"**:

Desligar&

Variável que possivelmente contém o que se procura

Informação procurada

Função que procura por uma informação específica

Índice do caractere a ser apagado

Variável com os caracteres

Função que remove caracteres

Quando essa palavra fosse ser analisada na linha 24, o ***indexOf()*** iria encontrar o que procura e o programa iria sempre enviar *HIGH* para o pino 8 do Arduino, o que impediria o LED de ser desligado. Por tanto, para evitar esse problema, decidimos usar ***"des&"*** para ser enviado como comando que irá desligar de fato o LED.

Com isso entendemos todo o processo para que o LED seja ligado a partir do momento em que o botão S1 do transmissor for pressionado e toda essa análise é válida para quando o botão S2 do transmissor for pressionado, ou seja, quando o transmissor enviar o comando ***"des&"*** o receptor agirá da maneira explicada, porém, agora quando o ***indexOf()*** da linha 28 encontrar o que procura, o programa irá enviar *LOW* para o pino 8, desligando o LED.

Funcionamento

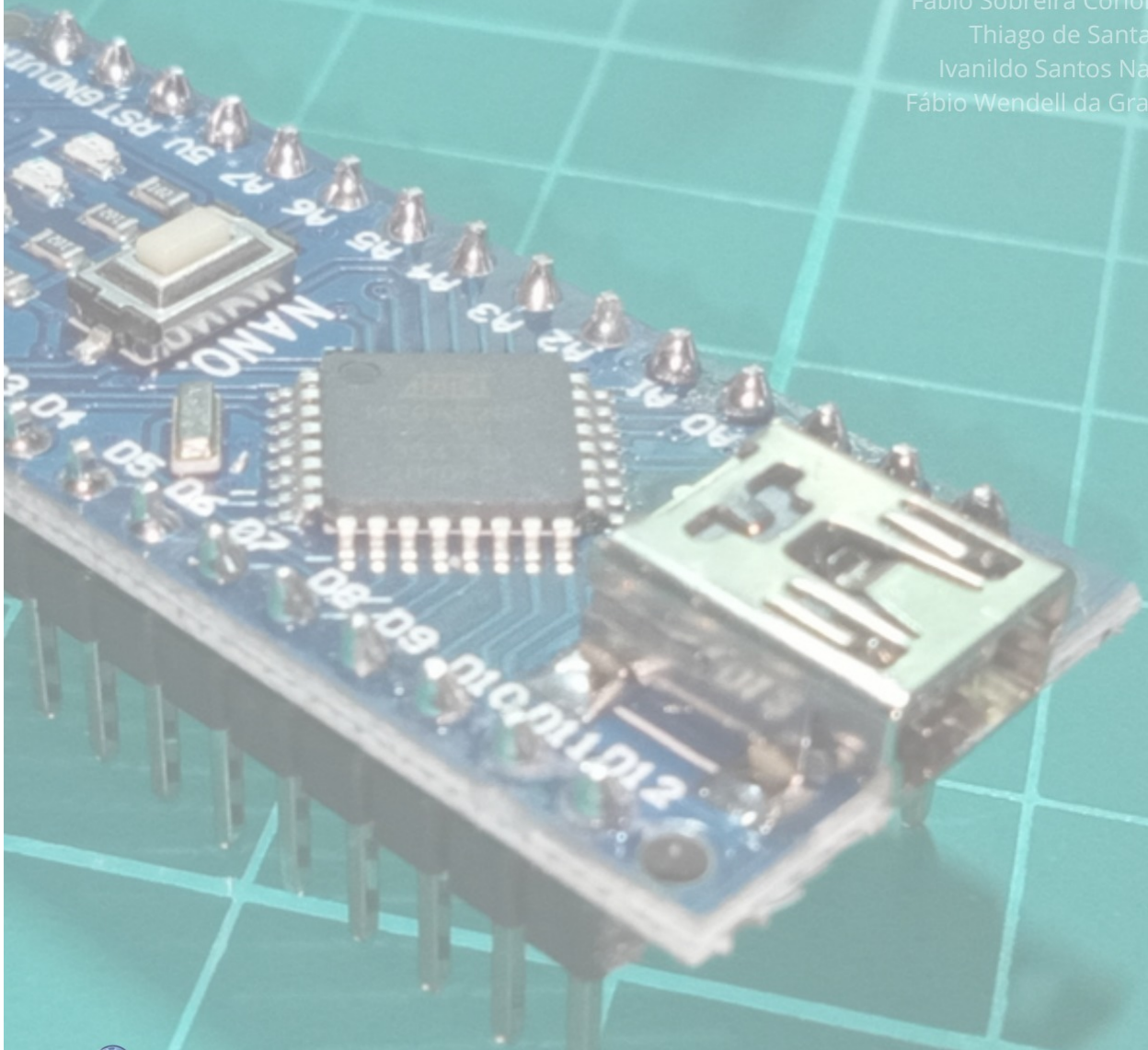
Depois de ter carregado os códigos no Arduino, ligue cada placa em uma porta USB diferente ou em computadores diferentes. Inicialmente o LED deverá iniciar apagado, mas ao pressionar o botão S1 ele deverá acender e assim permanecer e quando for pressionado o botão S2, o LED deverá apagar e assim se manter.

Uma sugestão interessante é usar duas baterias de 9V com clips, sendo uma para cada Arduino, conectadas como mostra a imagem abaixo para que assim os circuitos fiquem independentes de computadores:

Figura 55 - *Conexão da bateria de 9V ao Arduino para ter independência de computadores.*

ARDUINO ESSENCIAL:
PROJETOS PRÁTICOS PARA INICIANTES

Vanderlei Alves Santos da Silva
Diego Lopes Coriolano
Fábio Sobreira Coriolano Filho
Thiago de Santana Souza
Ivanildo Santos Nascimento
Fábio Wendell da Graça Nunes



conhecimentolivre.org/home
contato@conhecimentolivre.org
editoraconhecimentolivre

EDITORA CONHECIMENTO LIVRE