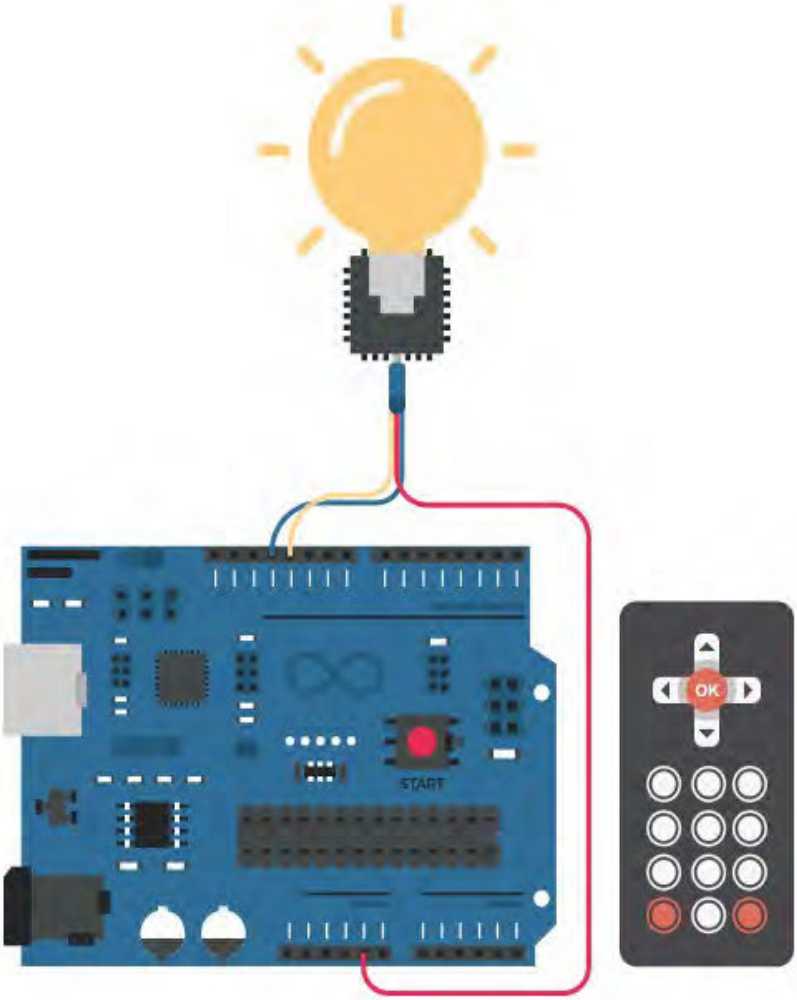
Arduino prático

10 projetos para executar, aprender, modificar e dominar o mundo



Casa do Código

FERNANDO BRYAN FRIZZARIN

© Casa do Código

Todos os direitos reservados e protegidos pela Lei nº9.610, de 10/02/1998.

Nenhuma parte deste livro poderá ser reproduzida, nem transmitida, sem autorização prévia por escrito da editora, sejam quais forem os meios: fotográficos, eletrônicos, mecânicos, gravação ou quaisquer outros.

Edição

Adriano Almeida
Vivian Matsui

Revisão

Bianca Hubert
Vivian Matsui

Revisão técnica

Carlos Panato

[2016]

Casa do Código

Livros para o programador

Rua Vergueiro, 3185 - 8º andar

04101-300 – Vila Mariana – São Paulo – SP – Brasil

www.casadocodigo.com.br

ISBN

Impresso e PDF: 978-85-5519-216-6

EPUB: 978-85-5519-217-3

MOBI: 978-85-5519-218-0

Você pode discutir sobre este livro no Fórum da Casa do Código: http://forum.casadocodigo.com.br/.

Caso você deseje submeter alguma errata ou sugestão, acesse http://erratas.casadocodigo.com.br.

AGRADECIMENTOS

A Deus.

À Priscila, minha esposa, pelo incentivo e imensa ajuda.

Aos meus filhos, Eduardo e Bruno, pelas inúmeras ideias.

À Nathália Lustosa da Silva, João Gabriel Sousa da Cruz e Victor Badolato Athayde, pela mão.

À Profª. Vanessa Juliato e à Profª. Magda Silva Rizzeto, pelo incentivo e encorajadora introdução.

À Fundação Romi, pela oportunidade de que eu aprenda ensinando.

SOBRE O AUTOR

Fernando Bryan Frizzarin, natural de Americana, é técnico em Informática, bacharel em Ciência da Computação e Psicopedagogo, especialista em Redes de Computadores, e tem MBA em Gestão Estratégica de Negócios.

Autor do livro *Arduino: guia para colocar suas ideias em prática*, publicado pela Casa do Código, professor do ensino superior nas Faculdades Integradas Einstein de Limeira e supervisor de informática do Departamento de Água e Esgoto de Americana. Ele foi ainda professor do Magistério Secundário Técnico na Universidade Estadual de Campinas (UNICAMP) por 12 anos. Também é voluntário na Fundação Romi no Núcleo de Ensino Integrado, em Santa Bárbara d'Oeste (SP), como consultor para o ensino de robótica no Ensino Fundamental.

Coautor da patente *BR1020140270159/2014: Dispositivo automatizado de liberação controlada*, projeto desenvolvido em conjunto com os alunos Bianca de Mori Machado Amaral e Felipe Ferreira da Silva, incluindo apoio da Arqª Marylis Barbosa de Souza. Esse projeto foi desenvolvido nas disciplinas de Desenvolvimento de Projetos e Tópicos Avançados em Informática no Colégio Técnico de Limeira (UNICAMP), e o depósito feito por meio da Agência de Inovação da UNICAMP (INOVA).

COMO USAR ESTE LIVRO

Minha formação sugere que o correto é mostrar e ensinar as ferramentas, e desafiar as pessoas para que elas as usem a seu favor. Isso para explorar e construir o conhecimento, que pode vir com diferentes nuances, cores, tamanhos e formas, conforme a necessidade e a forma como cada um encara seus desafios.

Porém, depois do livro *Arduino: guia para colocar suas ideias em prática*, mesmo não faltando desafios e sugestões do que é possível realizar com todas as ferramentas apresentadas, muita gente me procura com a pergunta: *"professor, me dá uma ideia do que fazer?"*. Sempre relutei em, de imediato, começar a sugerir projetos, pois acredito no potencial de todos em inventar. Mas comecei a achar que o que faltava, então, era um "pontapé inicial", algo concreto com o que se pudesse começar, que servisse como o desafio inicial para que seja usado, aprimorado e modificado livremente.

Sendo assim, este livro é para ser usado como um pontapé inicial e como um desafiador para novas ideias, para que você possa ter um projeto preconcebido, com a montagem e programação detalhada passo a passo, e componentes e lógicas básicas para que se possa ir além até onde sua imaginação permitir. Sugiro, inclusive, que você monte os projetos, modifique, e convide ou desafie mais alguém a continuá-lo, a ajudá-lo no aprimoramento para que haja uma troca de ideias e visões que permita que todos os envolvidos evoluam e cresçam juntos.

Apesar de este livro ser orientado para *hobbistas* e curiosos, profissionais também podem utilizá-lo. Porém, ele não apresentará conceitos profundos de eletrônica que devem ser buscados em concomitância com a leitura e a construção de cada projeto.

O pré-requisito será ter noções do uso de Arduino, um pouco de eletrônica, programação em linguagem C para o Arduino, além de bastante vontade de seguir em frente, ir além e criar. Desta forma, meu primeiro livro, já citado, é um par perfeito para este.

PREFÁCIO

O Arduino é uma plataforma formada por um equipamento eletrônico e um ambiente de programação integrado (*Integrated Development Enviroment* — IDE) para prototipagem eletrônica e de software.

O equipamento eletrônico da plataforma Arduino consiste em uma placa de circuitos integrados, devidamente equipada com seus componentes eletrônicos, cujo componente central é um microprocessador do tipo AVR da Atmel. Os softwares que serão utilizados para programar o Arduino são livres e gratuitos.

Muito se tem discutido, recentemente, acerca do uso do Arduino em projetos de tecnologia, mas pouco sobre como realmente utilizar esse componente, e o que de fato dá para se construir com ele. Este livro, obra preparada por Fernando Bryan, que ao longo de sua vida acadêmica vem criando, testando e inovando os projetos, nos traz muitos projetos com o Arduino. O material aqui apresentado tem como finalidade a apresentação de possíveis projetos que podem ser confeccionados com alguns componentes de elétrica.

Os projetos foram concebidos para usarem apenas componentes fáceis de encontrar em lojas especializadas de eletrônica, que tenham baixo custo e que possam ser, na medida do possível, compartilhados entre os projetos. No final do livro, você encontrará uma lista de todos os componentes utilizados em todos os projetos, indicando inclusive em quais projetos eles são utilizados.

Com linguagem simples, imagens com esquemas e links para vídeos, o presente livro lhe trará muitos projetos, e estimulará sua criatividade, explorando o seu raciocínio e possibilitando o

surgimento de novos ideários e como colocá-los em prática. Serão 10 capítulos com projetos prontos para seguir passo a passo, dando as orientações para desde automatizar uma porta (fazendo a abrir com senha), controlar a velocidade de um objeto com um radar, automatizar um sistema de iluminação através de controle remoto, construir um dado eletrônico, criar um videogame, entre várias outras ideias, até a construção de um robô aranha.

Este livro não foi escrito para quem quer aprender eletrônica, mas sim para quem quer aprender programar Arduino por meio destes projetos que não requerem exatamente muito conhecimento de eletrônica, mas sim de programação. Sendo assim, este livro é para quem sabe ou quer aprender programar, e se sabe programar — independe da linguagem ou sistema operacional — pelo menos intermediariamente.

Agradecemos ao Fernando Bryan, pelo o apoio e dedicação durante estes dois anos que vêm nos apresentando o Arduino e nos surpreendendo cada vez mais, não só com essa ferramenta tecnológica, mas com sua vontade de ensinar, empolgar e revolucionar os pensamentos e as ideias.

É com grande satisfação que apresentamos este livro, pois acreditamos no trabalho com o Arduino. Nós nos inspiramos muito com o livro *Arduino: Guia para colocar suas ideias em prática* em nossas aulas, e agradecemos a oportunidade de assim apresentar este segundo livro, que nos trará muitos projetos e novos conceitos de trabalhos. Além disso, vai estimular e envolver novos interessados nesse trabalho.

Siga em frente. É hora de aprender. Sinta-se à vontade!

Profª. Vanessa Juliato e Profª. Magda Silva Rizzeto

Fundação ROMI — Núcleo de Educação Integrada

Sumário

1 Como montar os projetos **1**

2 Projeto nº 01 — Criando nosso próprio Arduino **4**

2.1 Materiais utilizados nesse projeto 4

2.2 Desenvolvendo o projeto 5

2.3 Desafio 21

3 Projeto nº 02 — Automatizando uma porta com senha via teclado **23**

3.1 Materiais utilizados nesse projeto 23

3.2 Desenvolvendo o projeto 24

3.3 Desafio 43

4 Projeto nº 03 — Criando um radar para verificar a velocidade de um objeto **45**

4.1 Materiais utilizados nesse projeto 45

4.2 Desenvolvendo o projeto 46

4.3 Desafio 70

5 Projeto nº 04 — Que tal acionar as lâmpadas de sua casa com controle remoto? **71**

5.1 Materiais utilizados nesse projeto 71

5.2 Desenvolvendo o projeto 72

5.3 Desafio 86

6 Projeto nº 05 — Um dado eletrônico para jogos 88

6.1 Materiais utilizados nesse projeto 88

6.2 Desenvolvendo o projeto 89

6.3 Desafio 112

7 Projeto nº 06 — Criando um videogame para você 113

7.1 Materiais utilizados nesse projeto 113

7.2 Desenvolvendo o projeto 114

7.3 Desafio 128

8 Projeto nº 07 — Alarme de geladeira com monitoramento de abertura da porta 129

8.1 Materiais utilizados nesse projeto 129

8.2 Desenvolvendo o projeto 130

8.3 Desafio 147

9 Projeto nº 08 — Construindo um rastreador GPS offline 149

9.1 Materiais utilizados nesse projeto 149

9.2 Desenvolvendo o projeto 150

9.3 Desafio 178

10 Projeto nº 09 — Batata quente, quente... quente... queimou!

10.1 Materiais utilizados nesse projeto 180 **180**

10.2 Desenvolvendo o projeto 181

10.3 Desafio 188

11 Projeto nº 10 — O robô aranha mais simples do mundo! 189

11.1 Materiais utilizados nesse projeto 189

11.2 Desenvolvendo o projeto 190

11.3 Desafio 208

12 Lista geral de materiais 209

Capítulo 1

COMO MONTAR OS PROJETOS

Os projetos deste livro foram concebidos para serem montados usando equipamentos e componentes reais, sem o uso de simulador. Portanto, é importante obter todos os recursos antes de começar.

Serão usados apenas um Arduino e uma placa de ensaios, ou *prot-o-board* e vários outros componentes que são indicados e detalhados em cada projeto. Mas caso deseje mantê-los montados, você terá de providenciar um conjunto para cada projeto.

Você também precisará ter um computador para realizar a programação do Arduino, como também ter conexão à internet. A última versão do IDE Arduino pode ser baixado em http://arduino.cc. Note que é sempre uma boa ideia usar a versão mais atualizada.

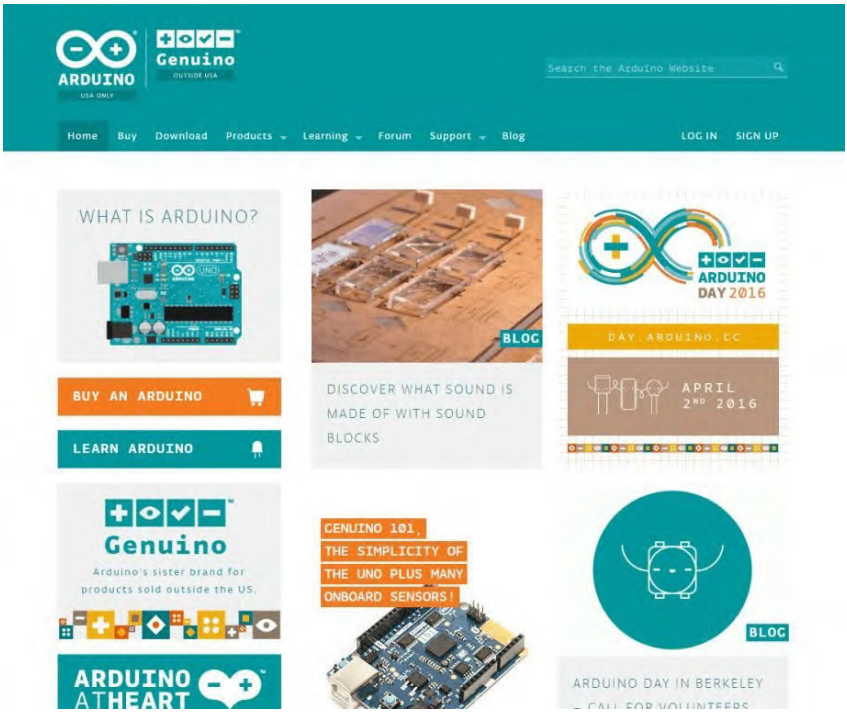


Figura 1.1: O site do Arduino (https://arduino.cc), acessado em 18 de março de 2016

Dê preferência por montar os projetos sobre uma mesa ou uma bancada. Depois de tudo pronto e funcionando, você poderá decidir sobre o acabamento, adaptações e instalação em local apropriado. Antes de começar, procure identificar corretamente os componentes e os esquemas de ligação.

Códigos-fonte

Todos os códigos-fontes apresentados nos projetos deste livro estão disponíveis em https://github.com/fbryan/fontes/.

As bibliotecas utilizadas para os projetos estão disponíveis em https://github.com/fbryan/, sendo que os caminhos diretos estão explicitados nos projetos.

Vale a pena sempre reforçar que esses código podem e devem ser melhorados. Estude-os, compreenda o funcionamento, modifique, melhore e faça adaptações, para que você possa usá-los em seus próprios projetos.

Agora que você já sabe do que precisará para começar, vamos começar e em grande estilo: que tal criar nosso próprio Arduino?

CAPÍTULO 2

PROJETO Nº 01 — CRIANDO NOSSO PRÓPRIO ARDUINO

Como prometido, vamos começar em grande estilo. Construiremos nosso próprio Arduino! Já pode ir pensando no nome que dará ao seu. Para o meu, chamarei carinhosamente de *Bryanduino*. Talvez seja falta de imaginação, eu sei, mas ficou simpático: "Braianduíno".

2.1 MATERIAIS UTILIZADOS NESSE PROJETO

- 1 x Arduino UNO R3
- 1 x Prot-o-board
- 1 x Suporte para baterias de 9V
- 1 x Bateria de 9V
- 1 x Cristal oscilador de 16MHz
- 1 x Resistor de 10 KΩ 1/4W
- 1 x Resistor de 330 Ω 1/4W
- 2 x Capacitores cerâmicos de 22 pF/50

- 1 x Regulador de tensão 7805
- 1 x LED vermelho
- Fios diversos

2.2 DESENVOLVENDO O PROJETO

Bom, mas saiba que não será a placa completa; não será esse o nosso objetivo. Vamos montar apenas o necessário para que o microcontrolador funcione corretamente e possa rodar nossos programas.

A ideia é que usemos um Arduino completo — desses que compramos em lojas ou na internet —, para prototipar nosso projeto, testar os componentes e a programação. Depois retiramos o microcontrolador da placa e colocamos no circuito básico, junto com os componentes e shields que, por ventura, sejam usados no projeto, de forma que ele possa funcionar compacto e apenas no que interessa.

Isso tudo para que possamos integrar nossos projetos em locais menores e melhores. Você verá que, em todo projeto, a sugestão é sempre que você faça e teste tudo usando a placa Arduino, mas depois construa seu próprio circuito para deixar o projeto menor e mais cômodo. Esse também é um bom começo, um jeito de criar uma relação mais íntima com o Arduino e suas possibilidades.

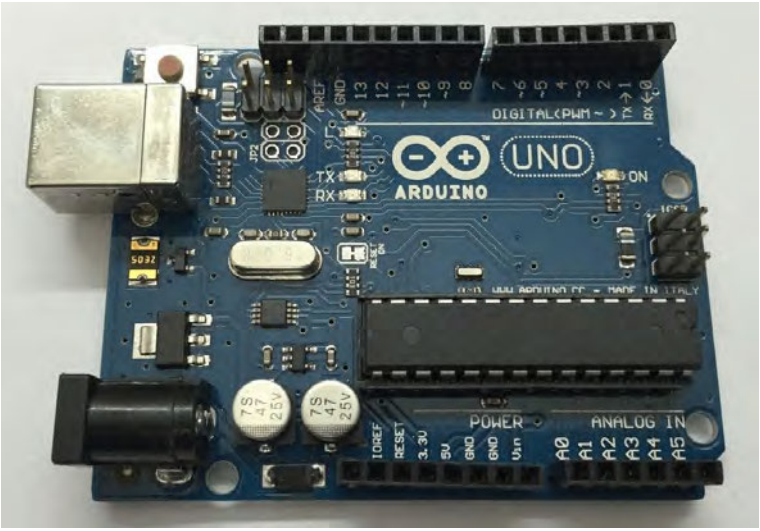


Figura 2.1: O Arduino UNO R3, de onde sairá o microcontrolador

Vamos começar com um programa simples, o mais simples de todos: piscar o LED do pino 13.

```
void setup() {
    pinMode(13,OUTPUT);
}

void loop() {
    digitalWrite(13,HIGH);
    delay(1000);
    digitalWrite(13,LOW);
    delay(1000);
}
```

Compile e carregue o programa no seu Arduino e verifique se tudo funciona bem. Ou seja, se o LED do pino digital 13 fica aceso por um segundo e, em seguida, apagado também por um segundo, piscando.

Desligue o Arduino desconectando-o do computador e, com muito cuidado, retire o microcontrolador da placa. Para isso, você pode usar uma pinça própria para remoção de circuitos integrados,

como mostrada na figura a seguir, ou uma chave de fenda bem fina.

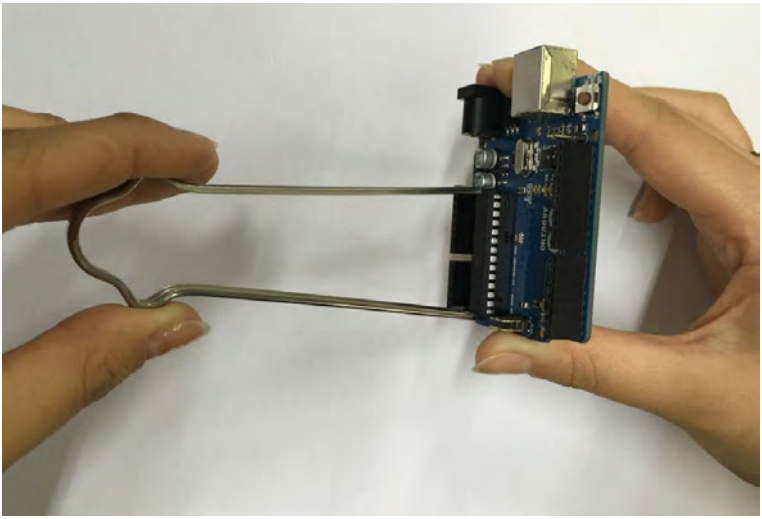


Figura 2.2: Removendo o microcontrolador da placa Arduino

Tenha o máximo de cuidado com a remoção do microcontrolador. Se você danificar qualquer pino, ele não funcionará nem fora, nem conectado de volta à placa do Arduino. Certifique-se, especialmente, de não entortar qualquer um dos pinos.

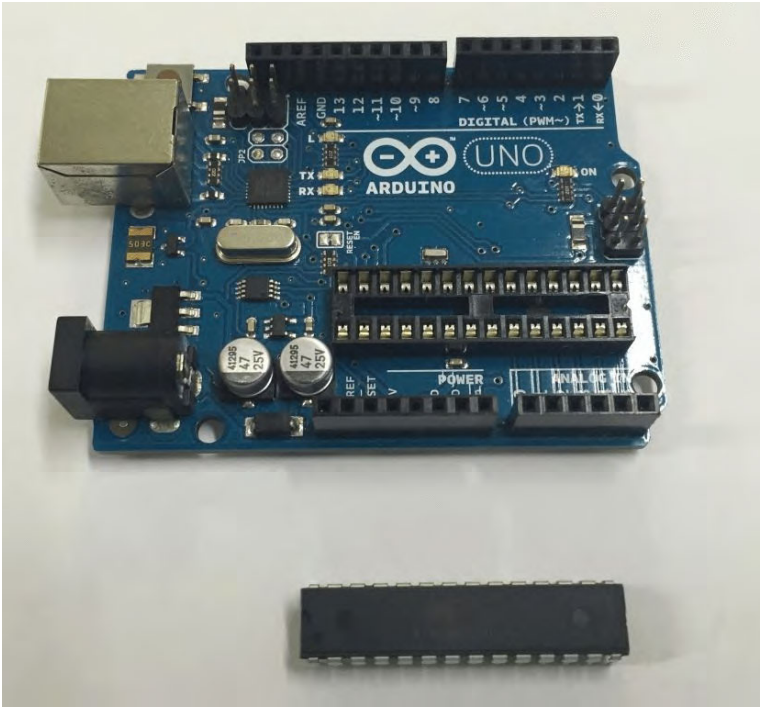


Figura 2.3: A placa do Arduino sem o microcontrolador e o microcontrolador fora da placa Arduino

Uma das coisas que você deve prestar muita atenção é na posição do microcontrolador. Note que, na parte superior, onde há as inscrições de marca e modelo (ATMEGA328), há também um chanfro e um ponto gravado em baixo relevo. Veja na figura a seguir:

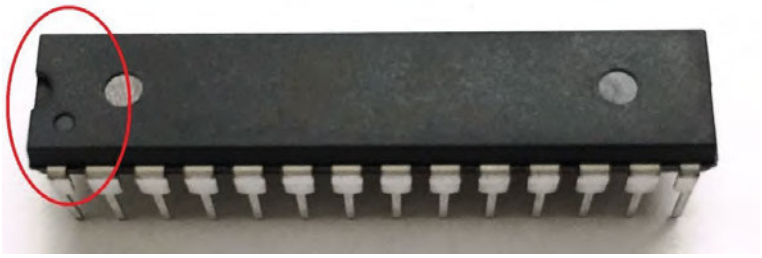


Figura 2.4: O microcontrolador com o chanfro e o ponto em baixo relevo em destaque à esquerda da imagem

Esse ponto indica que o pino mais próximo é o de número 1, e a contagem segue desse mesmo lado até o pino número 14. Do outro lado, o pino de menor número, o de número 15, é do lado oposto ao do ponto, ou seja, se o ponto está à esquerda, o pino 15 está do lado direito, e do outro lado, o microcontrolador. E desse outro lado, temos os pinos de número 15 até 28.

A figura seguinte ilustra esse esquema:

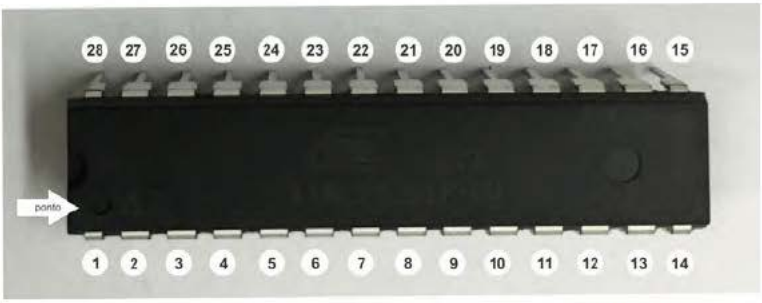


Figura 2.5: Os pinos, numerados, do microcontrolador ATMEGA328

A descrição dos pinos é o que segue:

- **Pino 1 — Reset**: colocar um nível baixo (GND) nesse pino causará o reset no microcontrolador, causando o reinício da execução do sketch a partir da função `setup()`.

- **Pino 2 — Serial RX (receiver/receptor) e digital 0**: pino digital 0 do Arduino que também é o pino serial RX nativo que pode ser usado para receber dados seriais de outro equipamento ou outro Arduino. Nesse outro equipamento ou Arduino, este deve estar interligado ao pino serial TX (transmissor).

- **Pino 3 — Serial TX (transmitter/transmissor) e digital 1**: pino digital 1 do Arduino, que também é o pino serial TX nativo, pode ser utilizado para enviar

dados seriais para outro equipamento ou outro Arduino. Nesse outro equipamento ou Arduino, este deve estar interligado ao pino serial RX (receptor).

- **Pino 4 — Pino digital 2**: pino digital 2 do Arduino.
- **Pino 5 — Pino digital 3 (PWM)**: pino digital 3 do Arduino, que é do tipo *Pulse-with Modulation* (pulso com modulação).
- **Pino 6 — Pino digital 4**: pino digital 4 do Arduino.
- **Pino 7 — VCC +5V**: pino para alimentação elétrica do microcontrolador com +5 volts.
- **Pino 8 — GND**: pino para conexão ao terra do microcontrolador (negativo).
- **Pino 9 — Cristal externo**: pino para conectar o microcontrolador ao cristal oscilador, também conhecido como *clock*.
- **Pino 10 — Cristal externo**: pino para conectar o microcontrolador ao cristal oscilador, também conhecido como *clock*.
- **Pino 11 — Pino digital 5 (PWM)**: pino digital 5 do Arduino, que é do tipo *Pulse-with Modulation* (pulso com modulação).
- **Pino 12 — Pino digital 6 (PWM)**: pino digital 6 do Arduino, que é do tipo *Pulse-with Modulation* (pulso com modulação).
- **Pino 13 — Pino digital 7**: pino digital 7 do Arduino.
- **Pino 14 - Pino digital 8**: pino digital 8 do Arduino.

- **Pino 15 - Pino digital 9 (PWM)**: pino digital 9 do Arduino, que é do tipo *Pulse-with Modulation* (pulso com modulação).

- **Pino 16 — Pino digital 10 (PWM)**: pino digital 10 do Arduino, que é do tipo *Pulse-with Modulation* (pulso com modulação).

- **Pino 17 — Pino digital 11 (PWM)**: pino digital 11 do Arduino, que é do tipo *Pulse-with Modulation* (pulso com modulação).

- **Pino 18 — Pino digital 12**: pino digital 12 do Arduino.

- **Pino 19 — Pino digital 13**: pino digital 13 do Arduino.

- **Pino 20 — AVCC +5V**: pino que provê alimentação elétrica para o conversor analógico/digital interno do microcontrolador, e deve ser conectado na alimentação de +5 volts.

- **Pino 21 — AREF**: pino que é a referência analógica para o conversor analógico/digital interno do microcontrolador. Pode ser alimentado com tensão de 0 até +5 volts.

- **Pino 22 — GND**: pino para conexão ao terra do microcontrolador (negativo).

- **Pino 23 — Pino Analógico 0**: pino analógico A0 do Arduino.

- **Pino 24 — Pino Analógico 1**: pino analógico A1 do Arduino.

- **Pino 25 — Pino Analógico 2**: pino analógico A2 do Arduino.

- **Pino 26 — Pino Analógico 3**: pino analógico A3 do Arduino.
- **Pino 27 — Pino Analógico 4**: pino analógico A4 do Arduino.
- **Pino 28 — Pino Analógico 5**: pino analógico A5 do Arduino.

É importante reconhecer os pinos corretamente para que seja possível você migrar seu projeto da placa Arduino completa para seu Arduino personalizado, obedecendo a ordem e esquema de ligações.

Além do microcontrolador, para construir um Arduino mínimo, você precisará de um prot-o-board. Ele não precisa ser exatamente como a da imagem a seguir, que será a base para esse projeto:

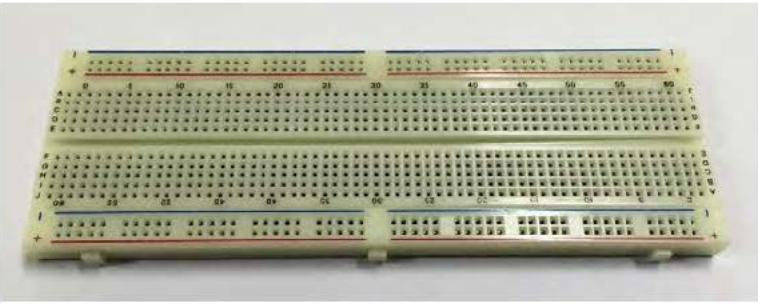


Figura 2.6: Uma prot-o-board

Você pode aproveitar e colocar o microcontrolador retirado do Arduino na prot-o-board:

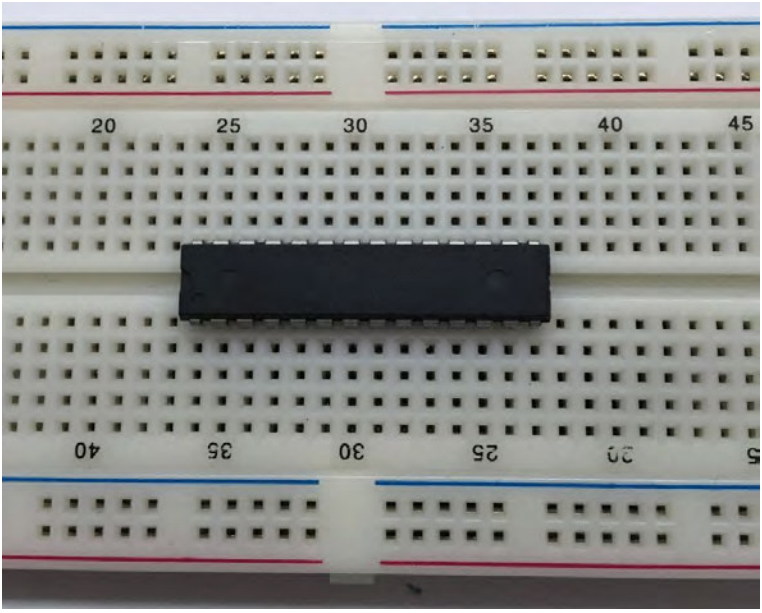


Figura 2.7: O microcontrolador ATMEGA328 conectado na prot-o-board

Você também precisará de um suporte para baterias de 9 volts para alimentar todo o circuito:

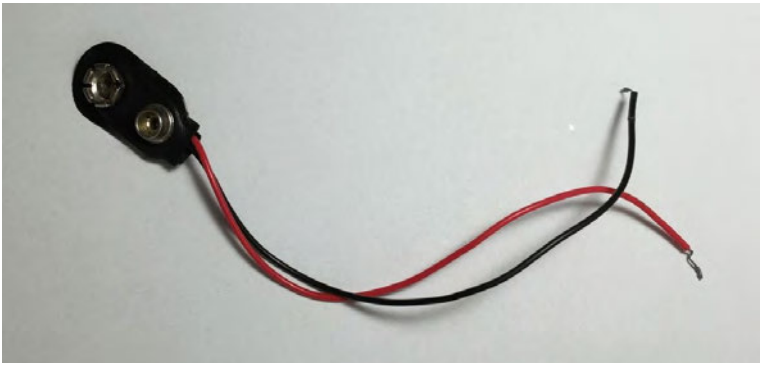


Figura 2.8: O suporte para baterias de 9 volts

Vamos aproveitar e também conectar o suporte para bateria na prot-o-board:

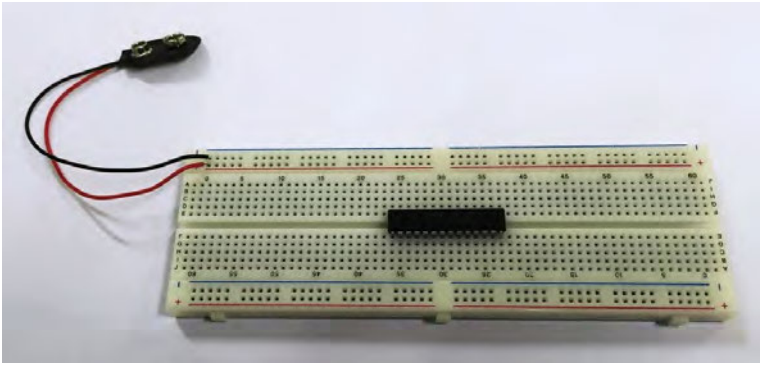


Figura 2.9: O suporte de bateria conectado às trilhas horizontais da prot-o-board

Um regulador de voltagem de, pelo menos, 20 volts para 5 volts. O utilizado será um transistor 7805, facilmente encontrado em lojas de materiais eletrônicos e com baixo custo.

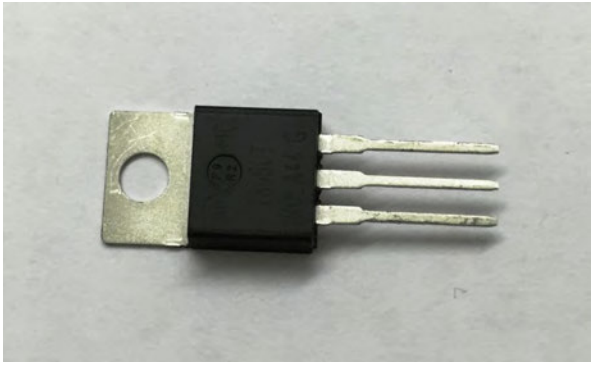


Figura 2.10: Um regulador de voltagem 7805

O transistor 7805 é um regulador de voltagem linear capaz de manter a tensão de saída constante em um determinado valor, no caso 5V, que é o requerido pelo microcontrolador. A tensão mínima de entrada é de aproximadamente 7V e máxima de 25V, com corrente máxima de 1,5A. Então, uma bateria de 9V dará conta do recado suficientemente e o circuito não vai requer mais do que isso em corrente.

Os três pinos são apresentados numerados na figura a seguir:

Figura 2.11: Um regulador de voltagem 7805 com os pinos numerados

O pino 1 é a entrada de tensão, conforme as especificações já citadas anteriormente. O pino 2 é o comum (ou seja, GND), e o pino 3 é a saída de tensão já regulada.

Então vamos colocar o regulador de tensão na prot-o-board, ligando o pino 1 do regulador ao positivo da bateria, o pino 2 do regulador ao negativo da bateria (GND) e o pino 3 do regulador ao pino 7 do microcontrolador (VCC +5V).

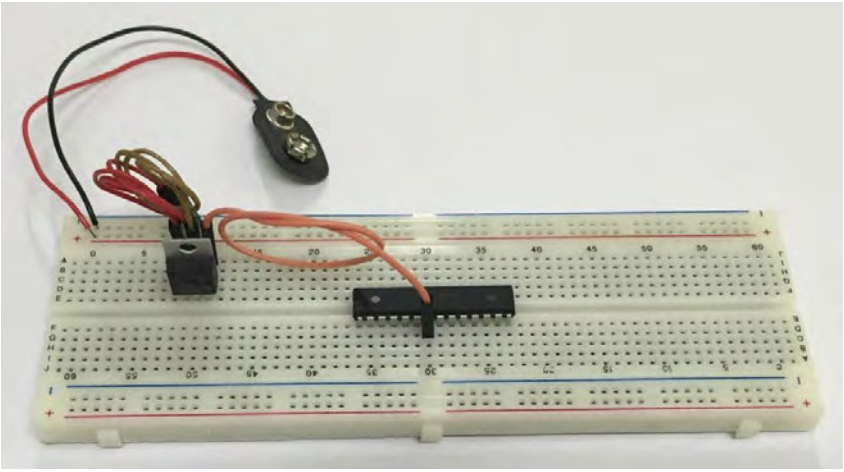


Figura 2.12: Conexões entre o regulador de voltagem, a bateria e o pino 7 do microcontrolador

Vamos aproveitar então para interligar o pino 20 do microcontrolador ao positivo da bateria, e os pinos 8 e 22 também do microcontrolador ao negativo da bateria (GND).

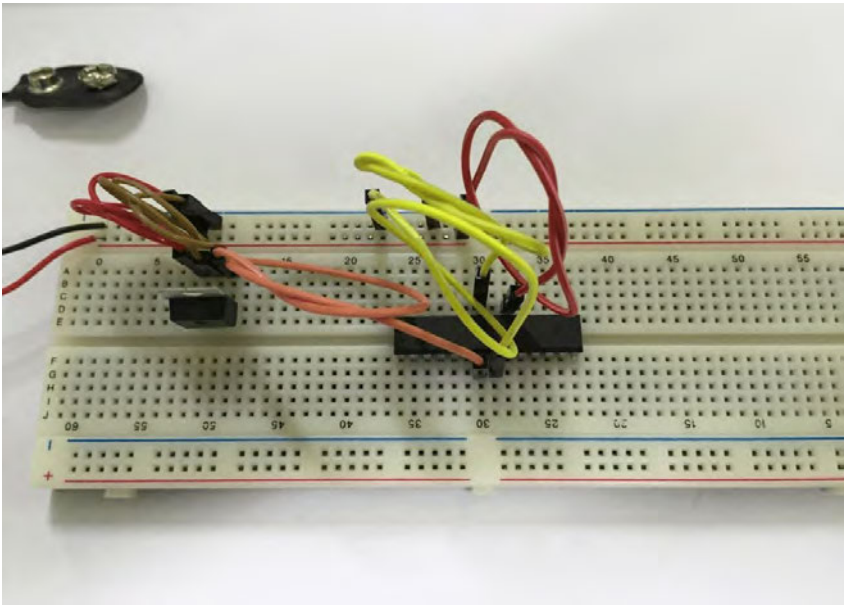


Figura 2.13: Os pinos 8 e 22 do microcontrolador ligados ao negativo e o pino 20 ligado ao positivo da bateria

O pino 21 do microcontrolador (AREF) também deve ser ligado ao positivo da bateria.

Precisamos também de um cristal oscilador para gerar frequência de clock, para que o microcontrolador possa funcionar.

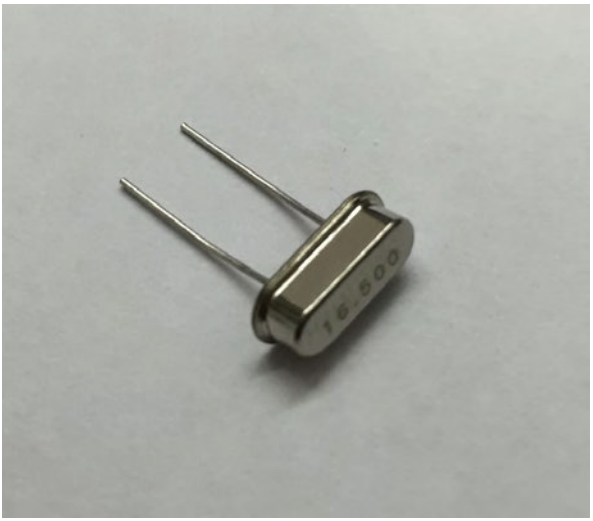


Figura 2.14: O cristal oscilador, usado como clock

O cristal oscilador será usado como gerador de sinal de clock para o microcontrolador. Ele é importante, pois garante o sincronismo e contagem de tempo para que o microcontrolador funcione corretamente.

Quando for adquirir um cristal, verá que há diversas frequências possíveis que são geradas por eles. O microcontrolador ATMEGA328 usado no Arduino UNO funciona em frequência máxima de 20MHz, mas, por padrão, nas placas Arduino UNO são usados cristais de 16MHz.

Ligue os pinos 9 e 10 do microcontrolador ao cristal oscilador. O cristal não possui polaridade.

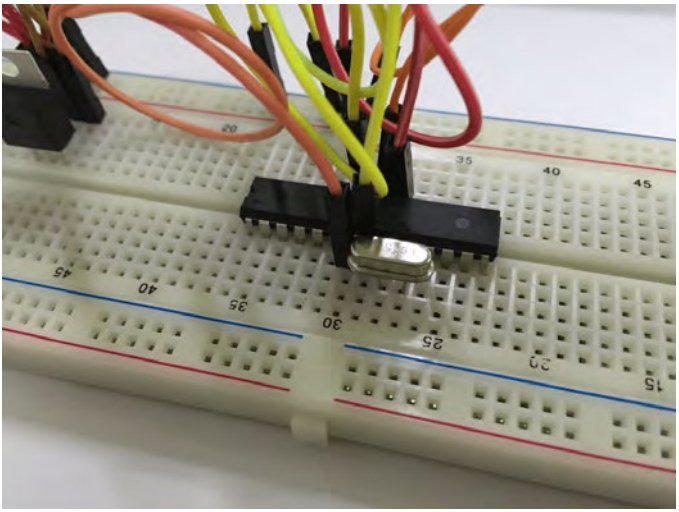


Figura 2.15: O cristal ligado ao microcontrolador

Falta pouco para terminar, apenas três componentes. Primeiro, dois capacitores cerâmicos de 22pF:

Figura 2.16: Os dois capacitores cerâmicos

Esses capacitores serão ligados cada um entre um terminal do cristal oscilador e o negativo da bateria (GND), como na figura a seguir:

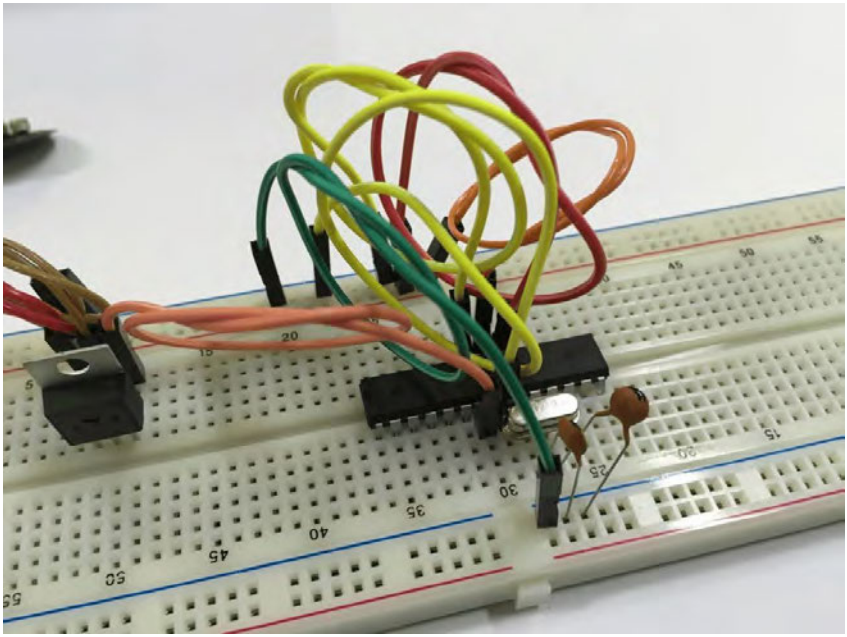


Figura 2.17: Capacitores entre os terminais do cristal oscilador e o negativo da bateria

O último componente é um resistor de 10KΩ, que será ligado entre a saída do regulador de tensão (+5V) e o pino 1 do microcontrolador. Como já explicado anteriormente, esse pino é o reset e, quando você colocá-lo um nível baixo (GND) nele, o microcontrolador será resetado e a execução do programa começará novamente. Portanto, com esse resistor, entre ele e o +5V, garantimos que ele terá sempre um nível alto (`HIGH`).

Caso queira incluir um botão para o reset, basta ligá-lo de forma que, quando pressionado, ligue o pino 1 do microcontrolador ao GND (negativo da bateria).

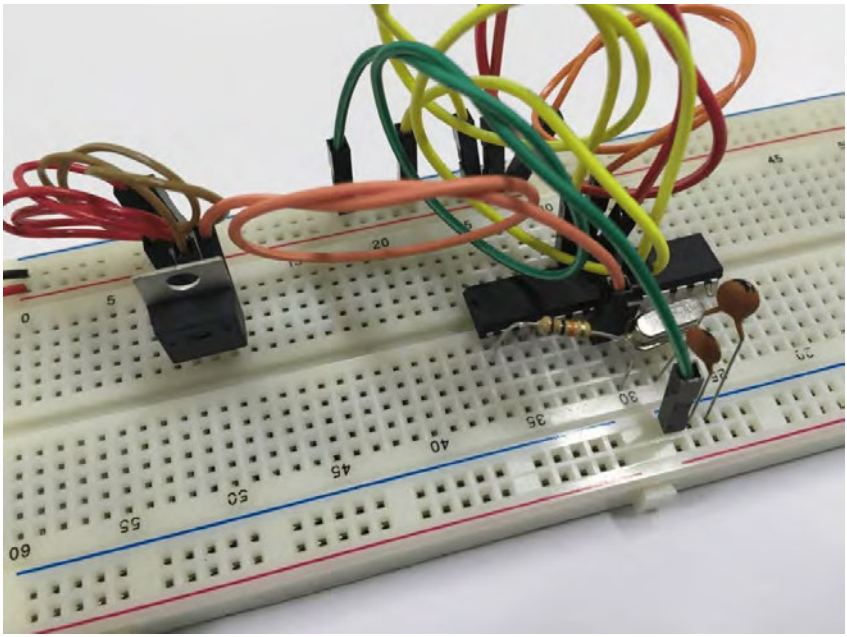


Figura 2.18: O resistor de 10KΩ entre o regulador de tensão e o pino 1 do microcontrolador.

Pronto! Aí está o seu Arduino montado na prot-o-board. Qual nome você dará? Agora, fiquei na dúvida se mantenho a ideia do Bryanduíno, ou se batizo ele de Fernanduíno. Mas o brinde em homenagem a ele está garantido.

Mas, e para testá-lo? Lembre-se de que programamos o microcontrolador no início desse projeto para piscar um LED que estiver ligado ao pino 13? Então, vamos ligar o conjunto LED e resistor no pino 13 do Arduino.

Ligue o resistor entre o pino 19 do microcontrolador (que é o 13 do Arduino) e o terminal positivo do LED. Ligue o terminal negativo do LED ao negativo da bateria (GND).

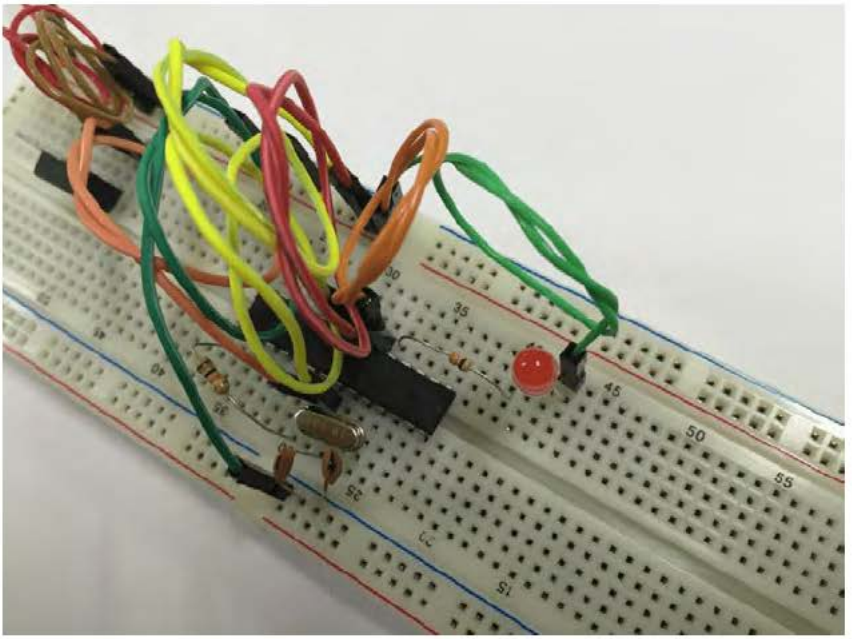


Figura 2.19: O conjunto LED e resistor no pino 13 do Arduino (19 do microcontrolador)

2.3 DESAFIO

Se você tiver boa imaginação ou criatividade, vai perceber que as possibilidades são infinitas. Você pode criar qualquer projeto, como faremos durante todo este livro. Pode testá-los usando uma placa Arduino UNO, e depois melhorá-los criando sua própria placa com o microcontrolador e o seu programa.

E esse será o grande desafio para todos os projetos que virão a seguir! Ele será sugerido em todos os projetos como você verá, além

de outras coisas não tão óbvias.

Mas para esse projeto em específico, um bom desfio será gravar o microcontrolador sem removê-lo da prot-o-board, e colocá-lo em uma placa Arduino UNO. A dica será que, para isso, basta usar um conversor USB/Serial e conectar os pinos RX serial do conversor ao TX serial do microcontrolador, o TX do conversor ao RX do microcontrolador e o GND do conversor ao negativo da pilha.

O próximo projeto pode ser o primeiro a receber o Arduino construído nele. Para que tudo caiba em uma caixinha que também pode conter o teclado, você poderá construir seu próprio Arduino para reduzir a necessidade de espaço dentro dessa caixa. Também será um projeto que pode ter utilidade prática e ajudá-lo a não ter mais que se lembrar das chaves para abrir portas e portões. Vamos lá!

CAPÍTULO 3

PROJETO Nº 02 — AUTOMATIZANDO UMA PORTA COM SENHA VIA TECLADO

Imagine que, como nos filmes de ficção científica, você chega até uma porta e não há maçaneta, nem mesmo fechadura, apenas um teclado ao lado. Nesse teclado, será possível destrancar a porta através da digitação da senha correta. Também deverá ser possível ajustar a senha através de uma contrassenha fixa, que será definida na programação do Arduino.

3.1 MATERIAIS UTILIZADOS NESSE PROJETO

- 1 x Arduino UNO
- 1 x Teclado matricial de membrana
- 1 x Transistor TIP 102
- 1 x Resistor de 1KΩ
- 1 x Relê 5V
- 1 x Fechadura elétrica com fonte 12V
- Fios diversos

3.2 DESENVOLVENDO O PROJETO

Nesse projeto, consideraremos que a sua porta já tenha maçaneta e fechadura, e que você não quer, necessariamente, trocá-la por uma nova. Portanto, a ideia será usar uma tranca elétrica comum, dessas utilizadas em portões e portas residenciais. Com isso, o custo será diminuído e a facilidade de adaptação será aumentada.

Vamos montar tudo, e depois você poderá decidir o melhor jeito de instalar onde desejar. Precisaremos de um Arduino UNO:

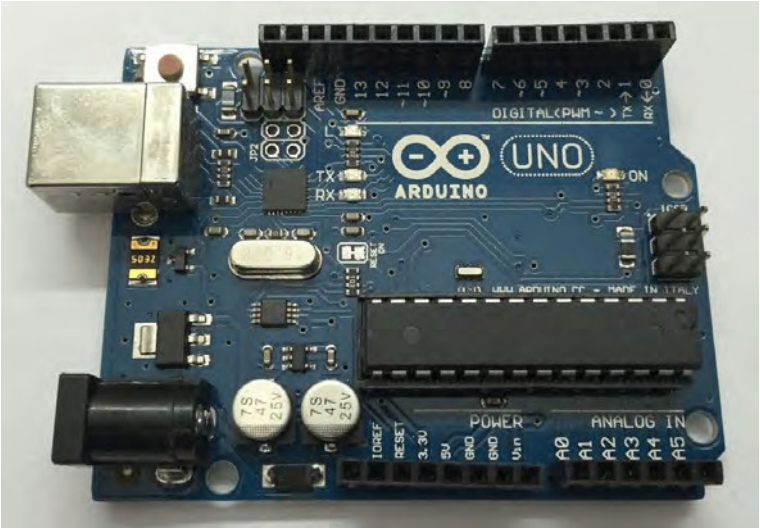


Figura 3.1: O Arduino UNO R3

Também será necessário uma prot-o-board para ser suporte de todo o circuito, que será muito pequeno:

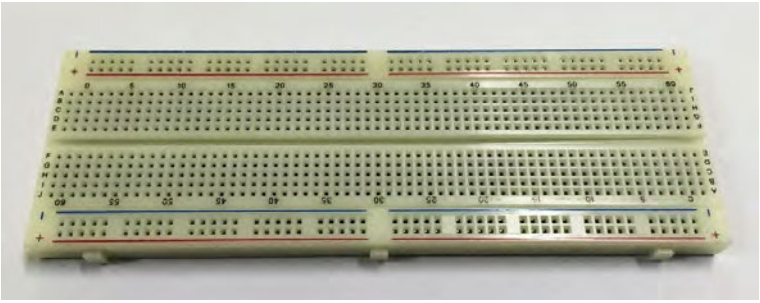


Figura 3.2: A prot-o-board utilizada, sendo que você pode usar qualquer uma que tiver disponível

Também precisaremos de um teclado matricial de membrana com, pelo menos, 12 teclas:

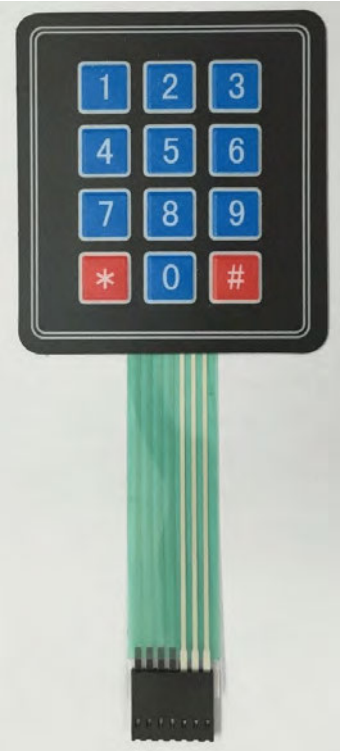


Figura 3.3: O teclado de membrana, apenas numérico, com 12 teclas, três colunas por quatro linhas

O teclado numérico de membrana com 12 teclas, como o mostrado na figura anterior, possui 7 fios, sendo que três são para acesso às colunas e quatro para as linhas. O pino 1 é o mais à esquerda, olhando-se o teclado de frente, e o pino 7 é o mais à direita.

O teclado ocupará a maioria dos pinos digitais do Arduino, já que cada pino do teclado terá de ter um correspondente no Arduino. Para capturar as teclas que serão pressionadas, usaremos a biblioteca `Keypad` que está disponível no endereço https://github.com/fbryan/keypad.

Nessa página, haverá um link para que seja feito o download da última versão da biblioteca. Esta trará algumas funções que mapeiam os pinos da matriz de botão que, na verdade, é esse teclado.

Para importar a biblioteca, abra o IDE Arduino e clique no menu `Sketch`. Em seguida, selecione a opção `Incluir Biblioteca`:

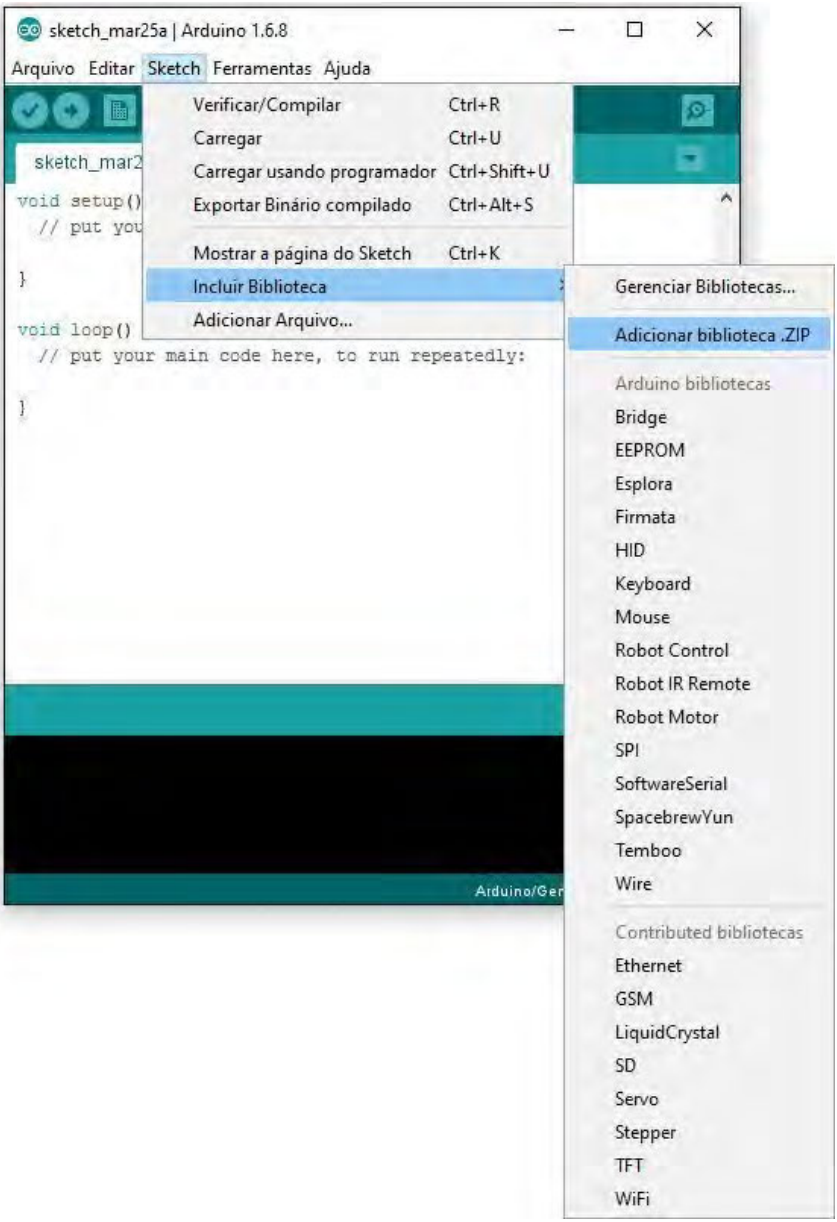


Figura 3.4: O menu Sketch do IDE Arduino com a opção Incluir Biblioteca selecionada.

Clique na opção `Adicionar biblioteca .ZIP` , que abrirá a seguinte caixa para seleção de arquivos:

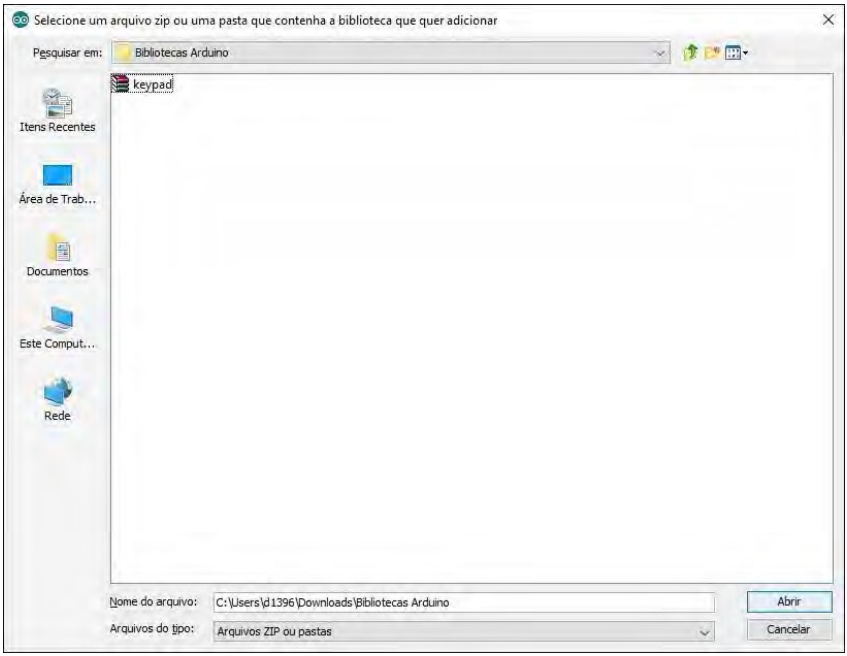


Figura 3.5: A janela para seleção da biblioteca keypad no formato .zip

Encontre onde você salvou o arquivo `keypad.zip` que obteve pelo download anterior. Depois, clique no botão `Abrir` . Pronto, a biblioteca foi adicionada. Se você abrir novamente o menu `Sketch` e selecionar a opção `Incluir Biblioteca` , veja que a Keypad já está disponível, como vemos a seguir:

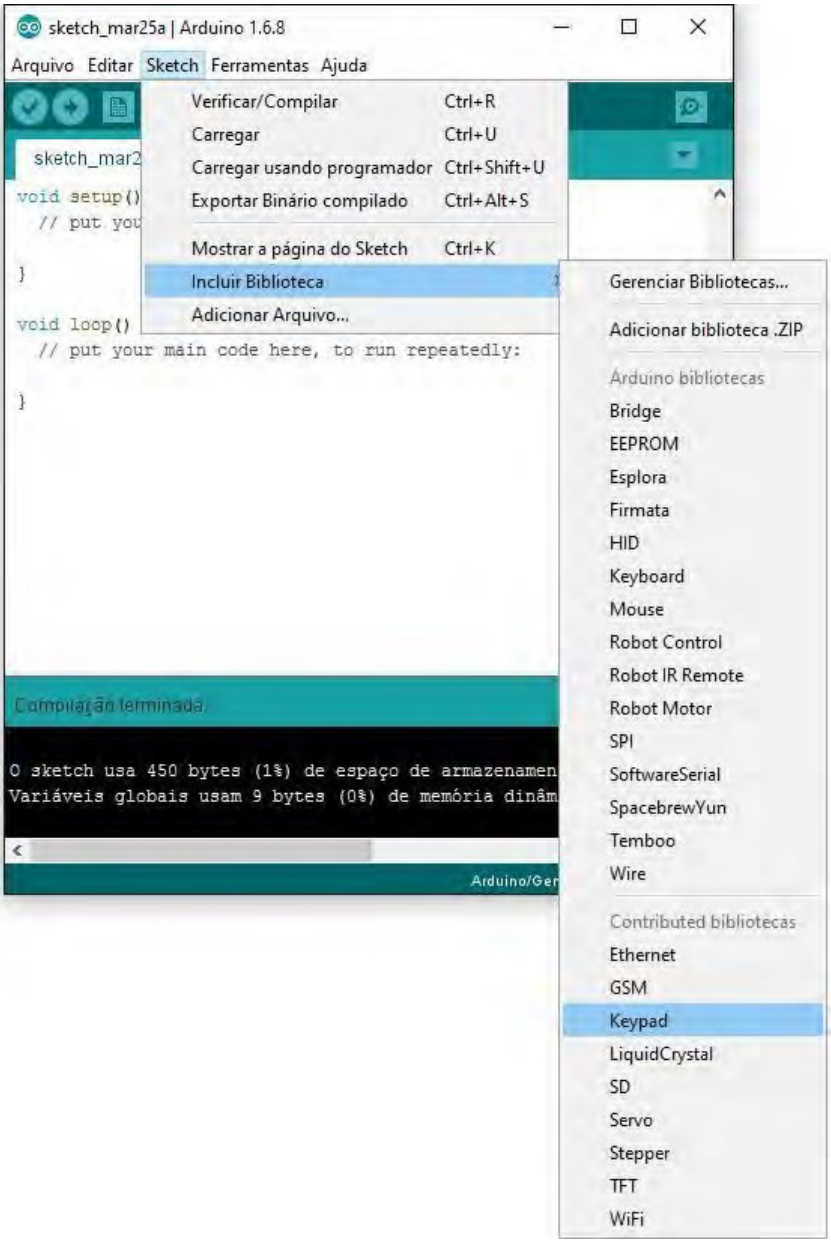


Figura 3.6: O menu Sketch com a opção Incluir Biblioteca selecionada. Veja a biblioteca Keypad disponível.

Já podemos começar a montar nosso projeto para valer, a partir

de agora. Vamos conectar e programar o teclado para já entender como ele funciona e como capturar o que o usuário vai digitar. Isso também adiantará parte do projeto.

Vamos conectar o teclado diretamente ao Arduino. Para isso, você precisará de 7 fios. Não farei a ligação via prot-o-board, porque não será necessário, e isso só aumentaria nosso uso de fios. Olhando-o de frente, começando da esquerda para a direita, ligue todos os pinos do teclado aos pinos digitais 9, 8, 7, 6, 5, 4 e 3 do Arduino, nessa sequência:

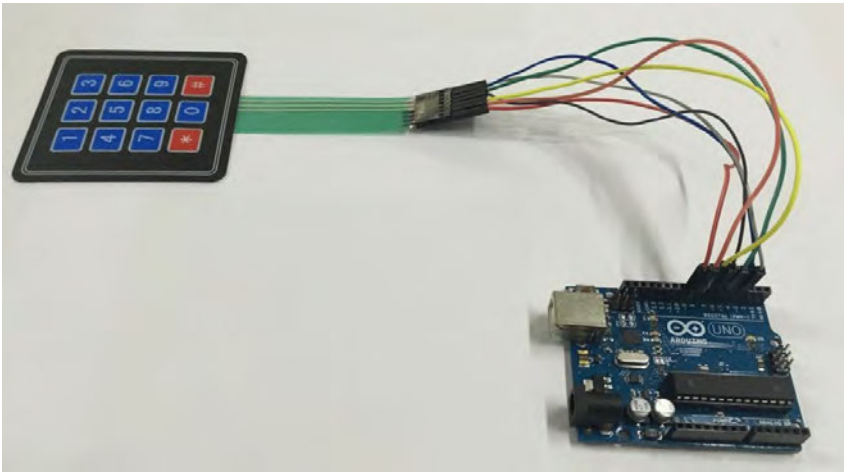


Figura 3.7: O teclado ligado aos pinos digitais do Arduino

Com tudo ligado, vamos para a programação:

```
#include <Key.h>
#include <Keypad.h>

const byte n_linhas = 4;
const byte n_colunas = 3;
char teclas[n_linhas][n_colunas] =    {
                        {'1','2','3'},
                        {'4','5','6'},
                        {'7','8','9'},
                        {'*','0','#'}
                    };
```

```
byte mapa_linha[n_linhas] = {9,8,7,6};
byte mapa_coluna[n_colunas] = {5,4,3};
Keypad teclado = Keypad(makeKeymap(teclas),mapa_linha,mapa_coluna,
n_linhas,n_colunas);

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

void loop() {
    char tecla = teclado.getKey();
    if(tecla != NO_KEY) {
        Serial.println(tecla);
    }
}
```

Com o auxílio da biblioteca `Keypad`, tudo fica mais fácil, mas vamos passo a passo. Os includes adicionam a biblioteca `Keypad` e seu complemento `Key` ao projeto. Essa biblioteca vai oferecer todos os objetos e métodos necessários para ler as teclas. Precisamos definir o número de linhas, que fazemos com a variável `n_linhas`, e também o número de colunas, feito com a variável `n_colunas`, que existem no teclado que vamos usar.

Os valores das teclas que existem são definidas na matriz `teclas`, que reflete a posição e quais são as teclas. Note que aí pode acontecer algo extremamente interessante: você pode definir quaisquer valores para as teclas. Se você quiser que quando o número um do teclado for pressionado seja lido o valor `%`, por exemplo, basta definir isso nessa matriz. O usuário achará que a tecla 1 envia um valor, mas na verdade não.

Usaremos, obviamente, os valores "normais" para as teclas, se não confundiria um pouco seus estudos. Mas essa é uma possibilidade que merece ser mencionada.

Com os vetores `mapa_linha`, definimos em quais pinos digitais do Arduino estão conectados aos pinos correspondentes às linhas do teclado. Fazemos o mesmo com os pinos correspondentes às

colunas do teclado com o `mapa_colunas`, como mostrado anteriormente na hora de conectar o teclado ao Arduino.

A linha `Keypad teclado = Keypad(makeKeymap(teclas),mapa_linha,mapa_coluna,n_linhas,n_colunas)` transforma todas essas definições anteriores no objeto chamado teclado, que é do tipo `Keypad`, e recebe a matriz com o mapa das teclas chamada `teclas`, os mapas de pinos para as linhas `mapa_linha` e `mapa_coluna`, para as colunas. Também recebe a quantidade de linhas e de colunas do teclado.

Na função `setup()`, iniciamos a conexão serial com 9.600 bauds com a linha `Serial.begin(9600)`. Já na função `loop()` é lida a tecla pressionada para a variável `tecla`, feito com a linha `char tecla = teclado.getKey()`. Depois, com a estrutura de seleção `if`, verificamos se alguma tecla foi pressionada com `if(tecla != NO_KEY)`. Essa estrutura fará com que, se uma tecla for pressionada, seu valor correspondente seja mostrado na saída serial com a linha `Serial.println(tecla)`.

Vamos testar o que temos até agora. Compile e faça upload do seu programa para o Arduino.

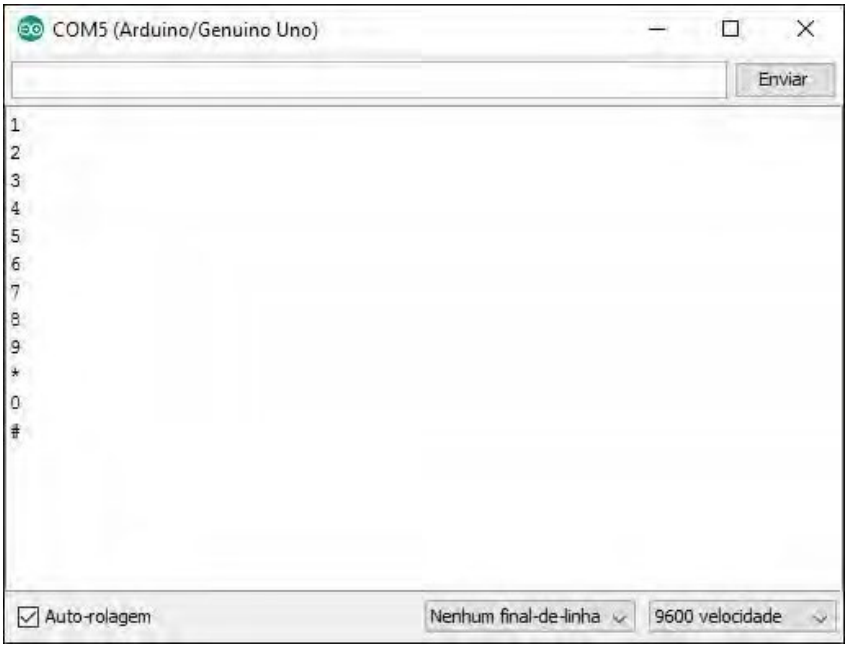


Figura 3.8: Serial monitor depois de pressionar todas as teclas

Com isso pronto, já temos o núcleo do nosso projeto quase concluído. Já que a ideia é ler uma senha, comparar com uma senha predefinida e liberar ou não a fechadura. Falta muito pouco, uma vez que controlar a fechadura será quase a mesma coisa que controlar um LED: ligado, libera a entrada; desligado, não libera.

Primeiro, seguindo essa lógica, vamos simular a fechadura elétrica com o LED do pino digital 13 do Arduino: digitaremos uma sequência numérica e compararemos com uma sequência predefinida. Caso coincidam, acederemos o LED; caso contrário, nada acontecerá.

Altere o código anterior para ficar como o código apresentado a seguir:

```
#include <Key.h>
#include <Keypad.h>
```

```
const byte n_linhas = 4;
const byte n_colunas = 3;
char teclas[n_linhas][n_colunas] =    {
                        {'1','2','3'},
                        {'4','5','6'},
                        {'7','8','9'},
                        {'*','0','#'}
                    };
byte mapa_linha[n_linhas] = {9,8,7,6};
byte mapa_coluna[n_colunas] = {5,4,3};
Keypad teclado = Keypad(makeKeymap(teclas),mapa_linha,mapa_coluna,
n_linhas,n_colunas);

int pos = 0;
int led = 13;
char senha[12];
char correta[] = "123456";

void setup() {
    pinMode(led,OUTPUT);
    Serial.begin(9600);
}

void loop() {
    char tecla = teclado.getKey();
    if(tecla != NO_KEY) {
        if(tecla != '#') {
            senha[pos] = tecla;
            pos++;
        }
        if(tecla == '#') {
            Serial.println(senha);
            if(!strcmp(correta,senha)) {
                Serial.println("Senha correta");
                digitalWrite(led,HIGH);
                delay(5000);
            }
            for(int i=0;i<strlen(senha);i++) senha[i] = ' ';
            pos = 0;
            digitalWrite(led,LOW);

        }
    }
}
```

O código não mudou muito em relação ao anterior, já que o uso das bibliotecas `Keypad` ficou inalterado. O que mudou foi que

criamos as variáveis `pos` que indicarão a posição de leitura da senha que está sendo digitada. Criamos também `led`, que indicará o pino digital do Arduino em que está o LED que acenderemos caso a senha esteja correta, no caso o pino 13. Já a variável `senha` receberá a digitação da senha e tem tamanho máximo de 12 caracteres, ou seja, limitamos o tamanho da senha em 12 números. Por último, criamos a variável `correta[]`, que contém a senha correta que deve ser usada para acender o LED.

Na função `setup()`, iniciamos o pino digital do LED como `OUTPUT`, ou seja, saída. Ainda continuamos usando a conexão serial para que possamos verificar o funcionamento correto do programa. Caso algo dê errado, você poderá conferir o que está acontecendo.

Na função `loop()`, lemos o que é digitado no teclado para a variável `tecla` e usamos a estrutura de seleção `if` para verificar se alguma tecla foi pressionada, exatamente como no primeiro código que fizemos. Depois disso, as coisas mudam um pouco.

Usamos uma estrutura de seleção `if` para verificar se a tecla digitada não foi `#` e, caso não tenha sido, armazenamos a tecla digitada na variável `senha` na posição definida pela variável `pos`, incrementada na sequência. Ou seja, cada dígito é armazenado na posição seguinte do vetor até o limite de tamanho determinado para a variável `senha`.

Se a tecla pressionada for `#`, enviamos a senha digitada para a conexão serial. Você poderá vê-la no Serial Monitor. Com outra estrutura `if`, comparamos a senha digitada com a esperada usando o comando `strcmp(correta,senha)`. Ele retornará `0` caso as variáveis tenham o mesmo conteúdo; um valor menor do que zero se a variável correta for menor que a variável `senha`; e um valor maior do que zero caso a variável correta seja maior que a senha digitada.

Tudo isso está na especificação do comando `strcmp` da linguagem C e causa um problema no `if` : zero é falso. Sendo assim, precisamos usar o operador `!` (`NOT`) para inverter o retorno deixando-o verdadeiro para as *strings* iguais.

Sendo a senha digitada correta, enviamos para a conexão serial a mensagem `Senha correta` com o comando `Serial.println("Senha correta")` , que você poderá conferir no Serial Monitor. O LED do pino digital 13 é aceso com `digitalWrite(led,HIGH)` e permanecerá assim por 5 segundos graças ao comando `delay(5000)` .

Na sequência, usamos `for(int i=0;i<strlen(senha);i++) senha[i] = ' '` para limpar a senha que foi digitada e aguardar uma possível nova digitação. Também zeramos a variável `pos` com `pos = 0` pelo mesmo motivo. Também temos de apagar o LED do pino 13; para isso, usamos `digitalWrite(led,LOW)` .

Com esse código, temos o projeto praticamente pronto! Percebeu? Temos apenas de trocar o LED do pino 13 por um relê, para que possamos controlar a fechadura elétrica com 110V usando o Arduino. Em vez de acender o LED, vamos acionar o relê que, por sua vez, ligará a fechadura e abrirá a porta.

Antes de continuar, lembre-se de tomar todo o cuidado necessário quando se utiliza a eletricidade. Siga todas as normas de segurança tal como realizar a montagem com todos os equipamentos desligados, proteger e isolar os contatos e não tocar partes metálicas e contatos enquanto o equipamento estiver em funcionamento. Os riscos são sérios e você deve realmente ter cuidado para não sofrer um acidente. Em caso de dúvida, reveja, reestude e procure ajuda.

Um relê é um interruptor eletromecânico que possui uma bobina com um núcleo metálico que é transformado em um eletroímã quando essa bobina está ligada. Quando o eletroímã, mostrado na próxima figura, entra em ação atrai contatos metálicos existentes dentro do relê realizando uma ligação elétrica entre eles. Há dois tipos de contatos metálicos em um relê: normalmente abertos e normalmente fechados que são movidos pela ação do eletroímã, como veremos já a seguir.

Figura 3.9: Um relê visto de baixo com seus terminais e a ligação da bobina em evidência

Nessa figura, os terminais numerados 1 e 2 são os contatos para acionar a bobina. O relê que vamos utilizar deve ter a bobina acionada por apenas 5V, para que possa ser controlado pelo Arduino. Assim com apenas 5V, poderemos acionar os contatos por onde passarão os 110V para acionar a fechadura elétrica.

Vejamos o esquema dos contatos:

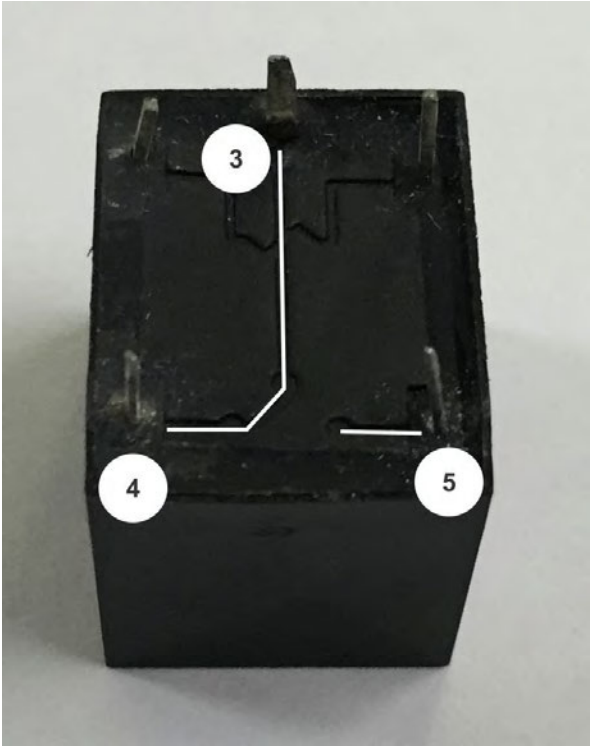


Figura 3.10: O relê visto de baixo com seus terminais e a ligação dos contatos em evidência

No caso dessa figura, os terminais que ligam a bobina não estão numerados, mas você já os conheceu antes. O conjunto formado pelos terminais 3 e 4 são os contatos normalmente fechados, e o conjunto de terminais 3 e 5 são os normalmente abertos.

Isso quer dizer que, com a bobina desligada, a ligação será entre o conjunto de terminais 3 e 4. E caso a bobina seja acionada, a ligação será entre o conjunto de terminais 3 e 5. E será assim que faremos nosso interruptor para a fechadura.

A figura a seguir mostra a posição dos terminais quando visto de cima para facilitar a visualização.

Figura 3.11: O relê visto de cima com os terminais numerados

Há um último detalhe antes de montarmos tudo. Não é possível ligar o relê diretamente ao pino digital do Arduino. A bobina do relê tem, em geral, resistência aproximada de 70Ω, e com voltagem de 5V que resulta na corrente de 0,07A — muito acima dos 0,04A tolerados pelo Arduino.

Porém, podemos acioná-lo pelo pino 5V, que está ligado diretamente ao regulador de tensão do Arduino que pode fornecer 0,5A. Como nosso circuito no total não consumirá corrente acima desse limite máximo, podemos usá-lo sem problemas. Para conseguir fazer tudo isso, vamos usar um transistor que fará a interface entre o pino digital do Arduino e o pino 5V para acionar a bobina do Arduino.

Primeiro, veja a ligação do transistor TIP102 na prot-o-board:

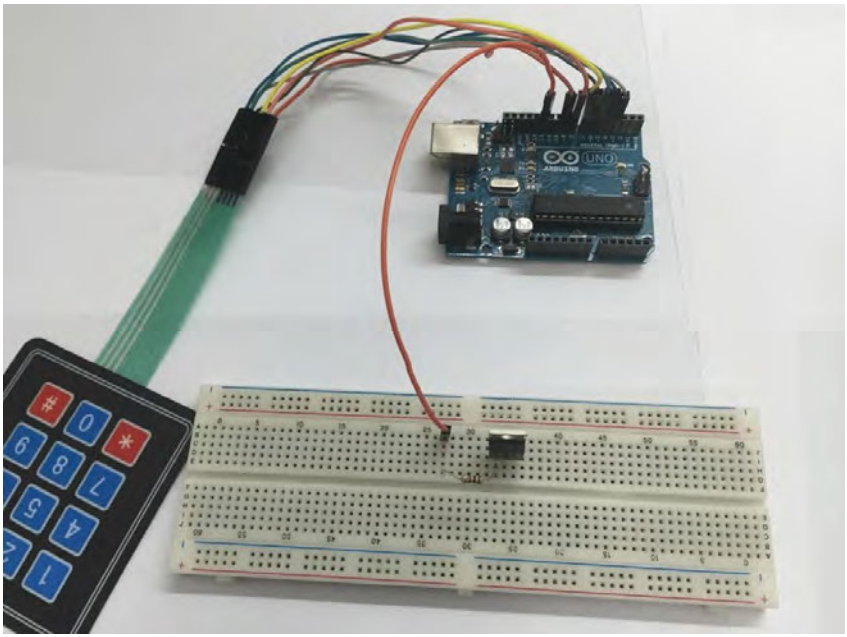


Figura 3.12: O transistor TIP102 ligado na prot-o-board e conectado ao Arduino

Conecte o terminal base do transistor a um resistor de 1KΩ, e o outro terminal do resistor ao pino digital 13 do Arduino. Assim, não mudaremos absolutamente nada no programa.

Ao terminal emissor do transistor, ligue um dos pinos GND do Arduino. O terminal coletor do transistor deve ser ligado a um dos terminais da bobina do relê. E o outro terminal da bobina do relê deve ser ligado ao 5V.

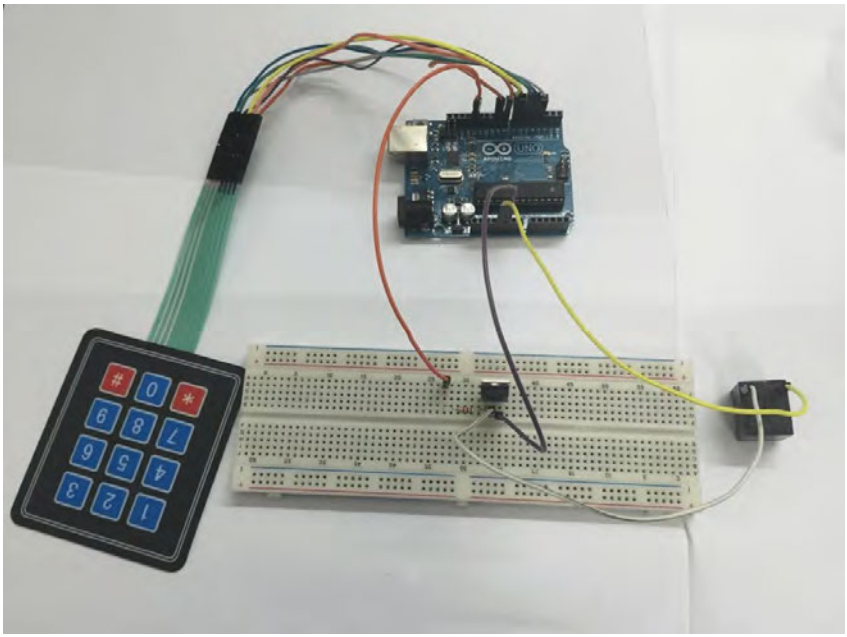


Figura 3.13: O transistor ligado ao GND do Arduino e à bobina do relê, e o relê ligado ao 5V do Arduino

Entre os contatos do relê normalmente aberto, vamos colocar nossa fechadura elétrica: basta secionar um dos fios e ligá-lo ao relê, nos contatos normalmente abertos. Assim quando acionarmos a bobina do relê, a fechadura também será acionada.

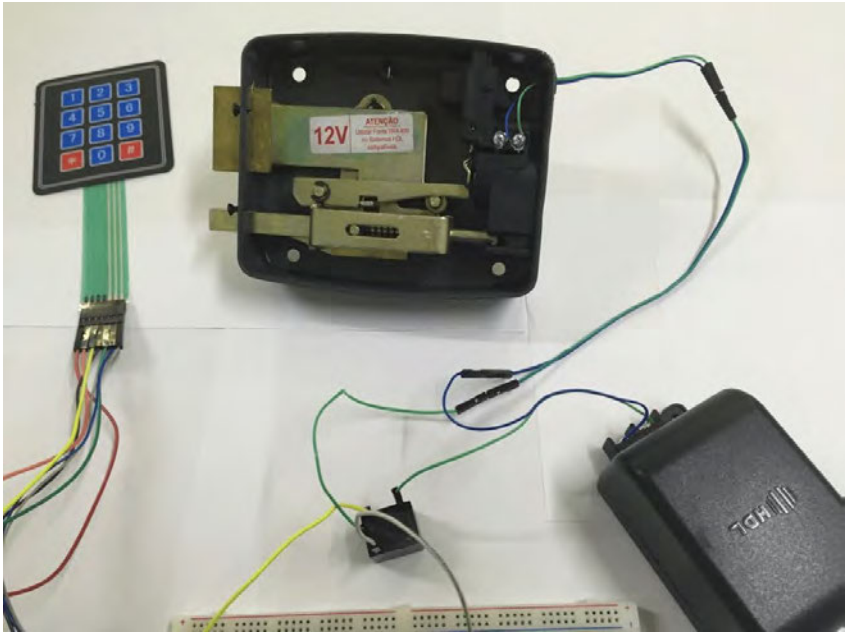


Figura 3.14: Um dos fios de acionamento da fechadura no relê secionado com o relê entre a fechadura e a fonte

A seguir uma figura panorâmica do projeto:

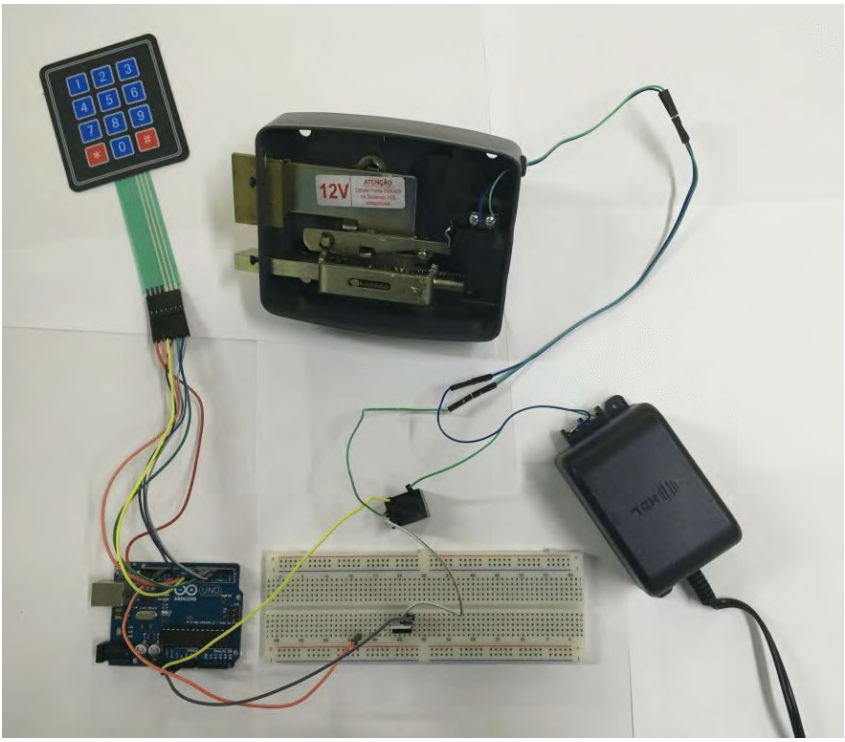


Figura 3.15: O projeto completo

Com tudo montado e pronto, basta digitar a senha correta e ver a fechadura ser acionada.

Um vídeo desse projeto em funcionamento pode ser visto em https://youtu.be/dR3-ssrLrWA.

3.3 DESAFIO

Incrementos para esse projeto não faltam. As possibilidades são praticamente infinitas!

Você pode incluir um buzzer e, a cada tecla, pode-se emitir um bip para indicar que a digitação aconteceu. Também é possível incluir um display LCD para indicar mensagens ao usuário, tal como "acesso negado", "digite sua senha" ou até mesmo "bem-vindo!".

Também é importante dar a possibilidade para o usuário gerar e gravar sua própria senha a partir de uma senha inicial. Para isso, você pode deixar a senha salva na memória EEPROM.

Na memória EEPROM, ainda você pode guardar quem abriu o portão (através da identificação da senha) e quando isso ocorreu. Depois, através de uma conexão serial (também pode-se pensar em bluetooth) com um computador, você pode fazer download desse log e saber como foi o trânsito nele local.

No próximo projeto, vamos medir a velocidade de qualquer coisa que se mova, mas relativamente de perto. Então, prepare-se e corra!

CAPÍTULO 4

PROJETO Nº 03 — CRIANDO UM RADAR PARA VERIFICAR A VELOCIDADE DE UM OBJETO

Com esse projeto, você pode ter horas e horas de puro deleite tentando encontrar quem entre seus amigos corre mais rápido, ou até criar pistas para "carrinhos de ferro" que permitam que eles atinjam altas velocidades.

Também pode verificar a velocidade de amigos em bicicletas ou carrinhos de rolimã. Só não pode se colocar em risco tentando quebrar recordes de velocidade. Lembre-se de sempre usar todos os equipamentos de segurança necessários, como capacete, joelheiras, cotoveleiras e outros aparatos para se proteger.

4.1 MATERIAIS UTILIZADOS NESSE PROJETO

- 1 x Arduino UNO
- 1 x Sensor ultrassônico
- 1 x Botão

- 1 x Resistor 1KΩ 1/4W
- 1 x Display de 7-segmentos com dois dígitos
- 2 x Resistores 330Ω 1/4W
- 1 x Suporte para baterias 9V com plug para o Arduino
- 1 x Bateria de 9V
- Fios diversos

4.2 DESENVOLVENDO O PROJETO

Antes de começarmos, preciso explicar que optei por não usar prot-o-board, pois quero deixar esse projeto pequeno e sem a necessidade de muito espaço, assim poderemos acomodar tudo em uma pistola daquelas que são usadas em videogames.

Então, vamos começar! Com o Arduino em mãos, você também precisará de um sensor ultrassônico. Esse sensor nos dá como retorno o tempo de viagem de uma onda sonora até um obstáculo, sua reflexão e posterior detecção. Como fazem os morcegos, por exemplo.

Veja a seguir uma figura do sensor com seus terminais numerados, que detalharei logo depois.

Figura 4.1: O sensor ultrassônico com seus terminais numerados

O terminal 1 deve ser ligado ao 5V e o 4 ao GND do Arduino, que são os terminais de alimentação elétrica do sensor. O terminal 2 é o gatilho (*trigger*) e deve ser colocado em alto (`HIGH`) e, logo em seguida, em baixo (`LOW`). Esse é o sinal para que o sensor comece a emitir os pulsos ultrassonoros e aguarde o retorno.

Depois que o gatilho é acionado, o sensor emite oito pulsos ultrassônicos (40 KHz), que não podem ser ouvidos por humanos por causa da frequência usada. Depois de emitir o pulso, ele deixa o terminal Echo (3) em alto (`HIGH`) até que esse pulso seja detectado depois de refletir (eco) em algum obstáculo, quando voltará o terminal Echo para `LOW` . Com a relação entre a velocidade do som e o tempo em que o terminal Echo ficou em `HIGH` , é possível determinar a que distância está o obstáculo.

Vamos colocá-lo para funcionar! Ligue o terminal VCC do sensor ao 5V do Arduino e, em seguida, o terminal GND do sensor a um dos GND do Arduino, como na figura a seguir.

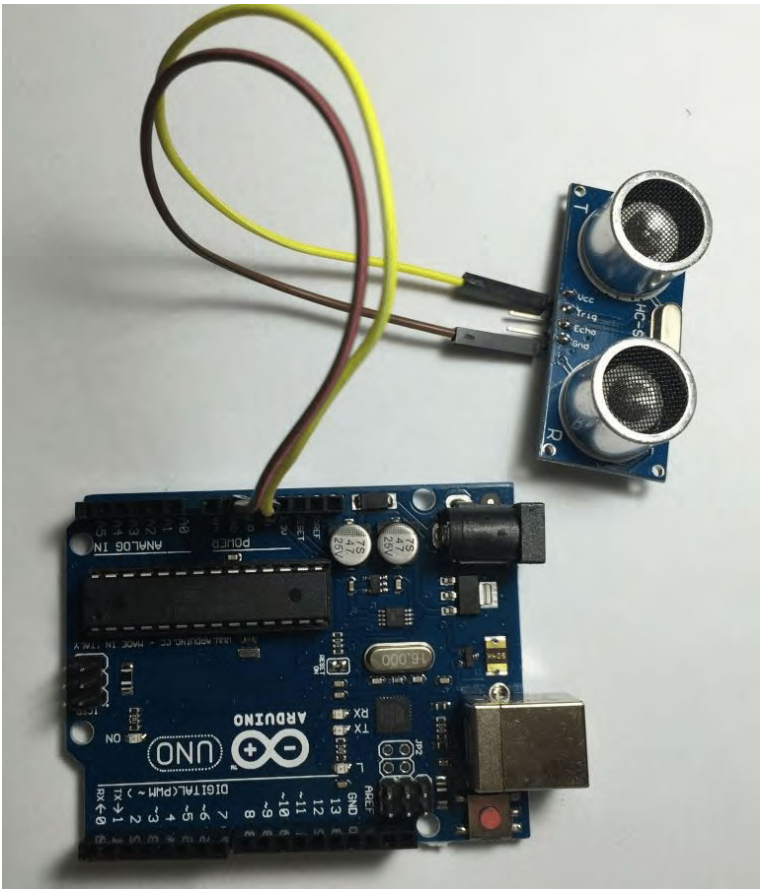


Figura 4.2: Terminais VCC (1) e GND (4) ligados ao Arduino

Agora ligue o terminal Echo ao pino digital 2 do Arduino, e o terminal Trig ao pino digital 3 do Arduino.

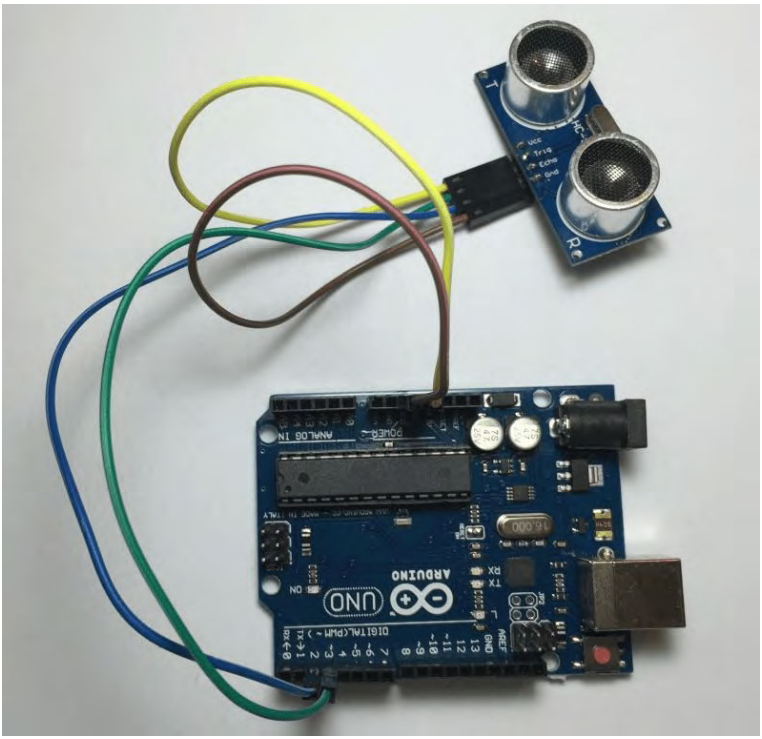


Figura 4.3: Terminais Echo (3) e Trig (2) do sensor ligados ao Arduino

Com as ligações feitas, podemos testá-lo. Antes de começar qualquer coisa, você precisará baixar e instalar no seu IDE Arduino a biblioteca Ultrasonic. Ela é superútil, pois nos poupa de ter que realizar vários cálculos para contar o tempo em que o pulso ultrassônico vai até um obstáculo e volta, e depois converter tudo em centímetros, metros ou até polegadas.

> Você pode fazer o download da biblioteca Ultrasonic em https://github.com/fbryan/ultrasonic.

Para instalar a biblioteca no seu IDE Arduino, use os mesmos

passos já apresentados no projeto nº 02 (*capítulo 03*).

Finalmente, vamos ao código:

```
#include <Ultrasonic.h>

Ultrasonic sensor(3, 2);

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

void loop()
{
  float medida;
  long tempo = sensor.timing();
  medida = sensor.convert(tempo, Ultrasonic::CM);
  Serial.print("Tempo de ida e volta = ");
  Serial.print(tempo);
  Serial.print(" microssegundos - ");
  Serial.print("Distancia medida = ");
  Serial.print(medida);
  Serial.println(" cm");
  delay(1000);
}
```

O primeiro passo é importar a biblioteca `Ultrasonic.h` com `#include <Ultrasonic.h>`. Já em seguida, criamos o objeto sensor como sendo do tipo `Ultrasonic`. Os parâmetros passados são onde estão ligados os terminais Trig e Echo, sendo que o Trig está no pino digital 3, e o Echo no pino digital 2 do Arduino.

Na função `setup()`, apenas iniciamos uma comunicação serial como computador em 9600 bauds, usando o comando `Serial.begin(9600)`. Na função `loop()`, declaramos uma variável do tipo `float`, chamada `medida`, que armazenará o que for medido pelo sensor.

Também declaramos uma variável do tipo `long` chamada `tempo`, que já é inicializada com o retorno do método `timing()` do objeto `sensor`, que é o tempo total que o pulso ultrassônico

leva para ser emitido, refletido em um objeto e ser detectado de volta. Esse método já inclui a rotina de disparar o gatilho do sensor e controlá-lo.

A variável `medida` recebe então o retorno do método `convert` do objeto `sensor`. Esse método deve receber como parâmetro microssegundos do tempo de ida e volta do pulso ultrassônico e a unidade de medida a ser usada. A medida a ser usada pode ser `Ultrasonic::CM` para retornar a medida em centímetros, ou `Ultrasonic::IN` para retornar em polegadas.

No restante do código mandamos os dados para o Monitor Serial e usamos um `delay(1000)` para fazer uma medição a cada segundo. Para testar, compile e envie o Sketch para o seu Arduino. Abra o Monitor Serial e você deverá ter como resultado algo semelhante à figura a seguir.

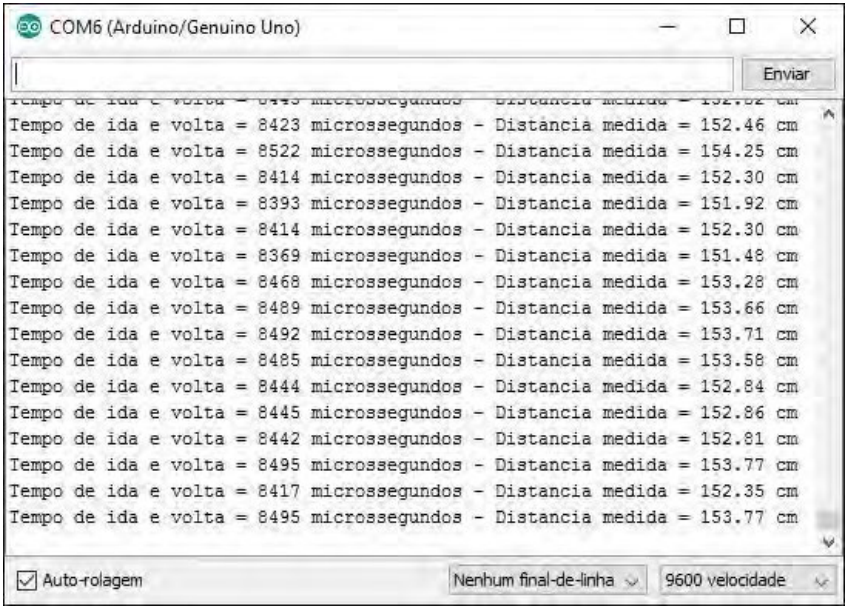


Figura 4.4: Monitor Serial apresentando o resultado das medições

Ok, sabemos como o sensor ultrassônico funciona e como programá-lo, mas ainda não é bem o que queremos. Precisamos fazer uma medição e, logo em seguida, fazer outra e, pela diferença de distância entre elas, calcular o deslocamento. Relembrando das aulas de Física do Ensino Médio e com o código a seguir, resolvemos isso:

```
#include <Ultrasonic.h>

Ultrasonic sensor(3, 2);

float anterior = 0;
float distancia = 0;
float deslocamento = 0;

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

void loop()
{
  long tempo = sensor.timing();
  float medida = round(sensor.convert(tempo, Ultrasonic::CM));
  distancia = abs(medida - anterior);
  deslocamento = (distancia * 10)/100;
  Serial.print("anterior = ");
  Serial.print(anterior);
  Serial.print(" - ");
  Serial.print("medida = ");
  Serial.print(medida);
  Serial.print(" - ");
  Serial.print("distancia = ");
  Serial.print(distancia);
  Serial.print(" - ");
  Serial.print("velocidade = ");
  Serial.print(deslocamento);
  Serial.println(" m/seg");
  delay(100);
  anterior = medida;
}
```

Esse código é basicamente uma modificação do anterior. Nele declaramos o objeto sensor do tipo `Ultrasonic`, como no código anterior, e três variáveis globais do tipo `float`.

A primeira variável é chamada `anterior`, que armazenará a medição imediatamente anterior, para que possamos fazer a relação entre duas medições e calcular o deslocamento, que também tem sua variável (`deslocamento`) para armazená-lo. A variável `distancia` armazenará a subtração entre a variável `anterior` (com a medição no instante anterior) e a variável `medida`, que contém a medição no instante atual.

Na função `setup()`, novamente iniciamos a comunicação serial com o computador com a taxa de transmissão de 9600 bauds. Já na função `loop()`, declaramos a variável do tipo `long` chamada `tempo`, que já é inicializada com o retorno do método `timing()` do objeto `sensor`, sendo o tempo total que o pulso ultrassônico leva para ser emitido, refletido em um objeto e ser detectado de volta. A variável `medida` dessa vez receberá o valor do método `convert` de `sensor` em centímetros já arredondado, para não conter muitas casas decimais.

Na sequência, calculamos a distância percorrida entre a medida anterior e a atual (variável `medida`). Para isso, basta subtrair uma da outra. Usamos a função `abs` para que retorne o valor em módulo, ou seja, ignorando o sinal. Esse resultado será armazenado na variável `distancia`.

Para saber o deslocamento, multiplicamos a distância percorrida por 10, pois queremos saber a medida em segundos, já que o tempo dado é 100 milissegundos entre as medições (comando `delay(100)` mais abaixo no código). Depois, precisamos dividir tudo por 100, uma vez que o desejado é obter os dados em metros e não centímetros, como o método `convert` do objeto `sensor` retorna.

A sequência de comandos `Serial.print` e `Serial.println` enviará todos os dados para o Monitor Serial, para que você possa acompanhar as medições. Para terminar, igualamos a variável

`anterior` à variável `medida`, e recomeçamos para fazer uma nova medição.

Depois de compilar e enviar o Sketch para o Arduino, abra o Monitor Serial e mova algum obstáculo, que pode ser sua mão, na frente do sensor. Você deverá ver alguma coisa muito similar ao mostrado na figura seguinte:

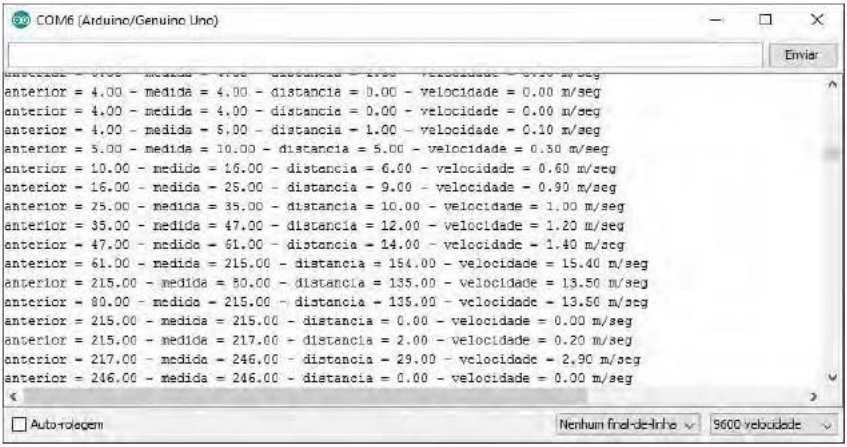


Figura 4.5: Resultado da medida de velocidade

Falta pouco! Na verdade, só mais três coisas:

- Um botão que sirva de gatilho para o radar de velocidade, que faça com que as medições possam ser controladas e não sejam contínuas.
- Um display de 7-segmentos com, pelo menos, 2 ou 3 dígitos para mostrar o resultado das medições.
- Uma bateria para alimentar tudo e deixar o projeto portátil.

Vamos pela ordem, primeiro: o botão. Ligar um botão ao Arduino é bem simples. Conecte um dos terminais do botão a um dos pinos GND, e o outro terminal ao pino digital 4 do Arduino.

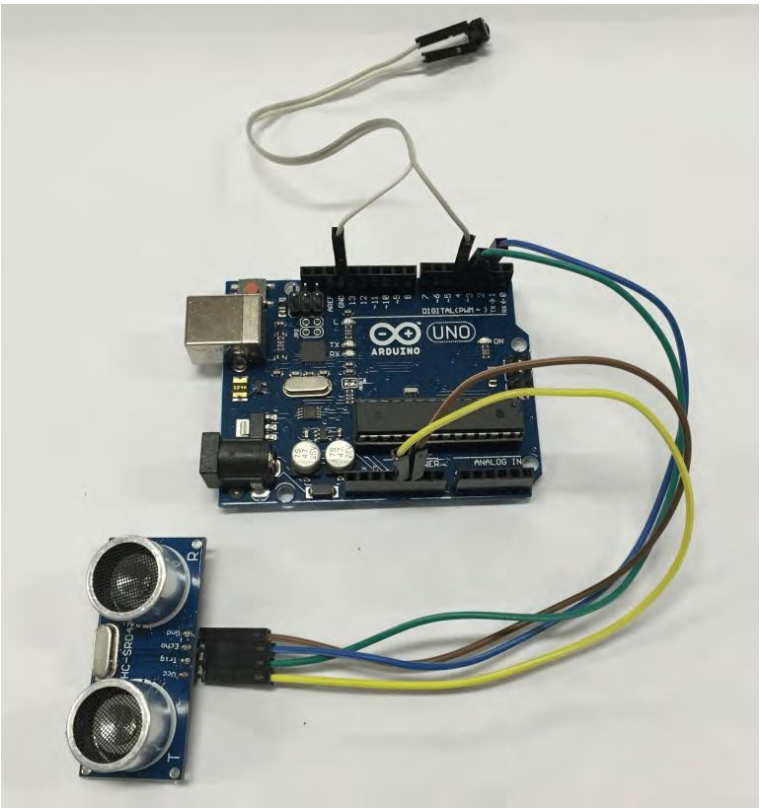


Figura 4.6: O botão ligado ao Arduino

Como não estamos usando prot-o-board, não temos como compartilhar o pino digital 4, nem o 5V com mais de um componente. Então, usarei o resistor *PULL-UP* interno do Arduino no pino digital 4.

O resistor PULL-UP interno do Arduino é acionado via software e possui valor entre 20KΩ e 50KΩ. Ele terá o efeito de garantir o funcionamento do botão. Ou seja, quando o botão estiver aberto, leremos no pino digital 4 o valor `HIGH`, e quando ele estiver fechado (pressionado), leremos o valor `LOW` nesse mesmo pino.

Vamos mudar nosso código para que apenas ocorram medições

quando o botão estiver pressionado:

```
#include <Ultrasonic.h>

Ultrasonic sensor(3, 2);

int botao = 4;

float anterior = 0;
float distancia = 0;
float deslocamento = 0;

void setup()
{
  Serial.begin(9600);
  pinMode(botao, INPUT_PULLUP);
}

void loop()
{
  if (digitalRead(botao) == LOW) {
    long tempo = sensor.timing();
    float medida = round(sensor.convert(tempo, Ultrasonic::CM));
    distancia = abs(medida - anterior);
    deslocamento = (distancia * 10) / 100;
    Serial.print("anterior = ");
    Serial.print(anterior);
    Serial.print(" - ");
    Serial.print("medida = ");
    Serial.print(medida);
    Serial.print(" - ");
    Serial.print("distancia = ");
    Serial.print(distancia);
    Serial.print(" - ");
    Serial.print("velocidade = ");
    Serial.print(deslocamento);
    Serial.println(" m/seg");
    delay(100);
    anterior = medida;
  }
}
```

As mudanças desse código para o anterior são realmente poucas outra vez: declarei uma variável inteira chamada `botao`, inicializada com o número 4, indicando em qual pino digital está ligado o botão. Com `pinMode(botao, INPUT_PULLUP)`,

inicializamos o pino do botão com a ativação do resistor PULL-UP interno, como explicado anteriormente.

Caso o botão esteja pressionado, o retorno do pino digital será `LOW`, então usei a estrutura de seleção `if(digitalRead(botao) == LOW)` para ler o retorno do pino e compará-lo com `LOW`. Caso a comparação seja verdadeira, será executado o código que realiza as medições do sensor ultrassônico e calcula a velocidade.

Só faltam duas coisas: incluir um display de 7-segmentos de dois dígitos ou dois displays de 7-segmentos de um dígito para mostrar a velocidade e colocar tudo em algum lugar para deixar mais prático do que a mesa. Então, vamos a ele.

Um display de 7-segmentos é um mostrador para números apenas (apesar de que gosto muito de inventar com ele, fazendo-o mostrar outras coisas).

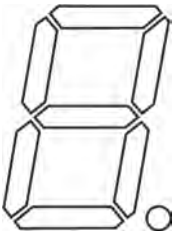


Figura 4.7: Um display de 7-segmentos

O display é formado por 7 LEDs que, acesos ou apagados em conjunto, mostram formas de números. Também há o ponto decimal, que também é um segmento que pode ser aceso ou não. Veja a figura a seguir:

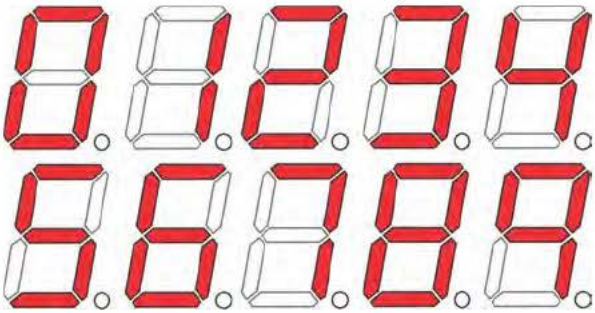


Figura 4.8: De zero até nove no display de 7-segmentos

Atrás do display, existem os terminais de ligação para podermos acender os segmentos, e dois tipos de displays:

- Ânodo comum, possuindo um pino que é o polo negativo comum a todos os segmentos que devem ser conectados ao polo positivo para serem acesos;
- Ou cátodo comum, possuindo um pino que é o polo positivo comum a todos os segmentos que devem ser conectados ao polo negativo para serem acesos.

Os segmentos são identificados de forma padronizada por letras de A até G, mais o DP (ponto decimal), que estão interligados cada um a um terminal correspondente atrás do componente. Além desses, há também o terminal comum, como explicado anteriormente.

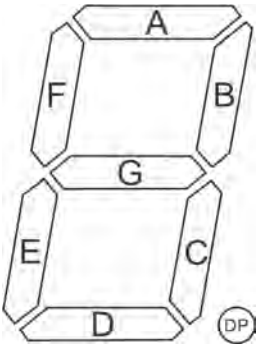


Figura 4.9: Os segmentos identificados

O display de 7-segmentos que usarei terá dois dígitos. Ou seja, será composto de dois display conjugados, sendo que cada terminal ligará os segmentos de ambos os display. Mas para que o segmento acenda, será preciso alimentar também o terminal comum, que no meu caso será ânodo (positivo).

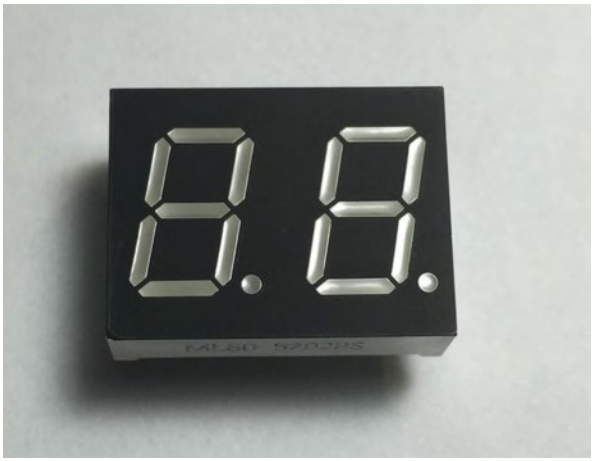


Figura 4.10: O display de 7-segmentos com dois dígitos

Os terminais estão ligados aos segmentos, ao ponto decimal e ao comum, conforme enumeração mostrada na figura a seguir que detalharei logo após:

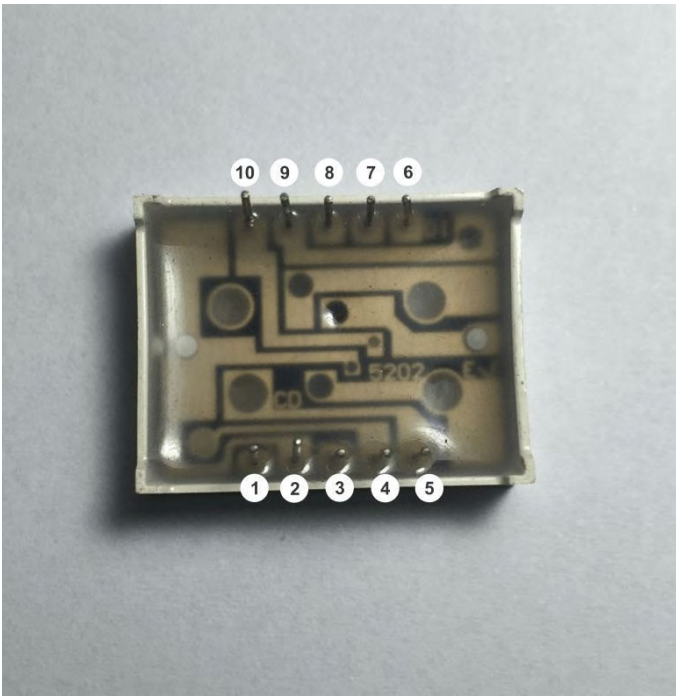


Figura 4.11: Os terminais do display numerados de 1 até 10

Os pinos dos segmentos são comuns aos dois dígitos. Para mostrar um número em apenas um segmento, basta acioná-lo através do pino ânodo comum do dígito desejado. Para mostrar números diferentes em cada um dos dois dígitos do display, basta acionar os segmentos de um dos dígitos por alguns milissegundos, depois acionar os segmentos do outro dígito também por alguns milissegundos e ir alternando entre eles. A mudança é tão rápida que, para o olho humano, parecerá que os dois dígitos estão acesos o tempo todo.

A sequência dos terminais é:

1. Segmento D
2. Ponto decimal
3. Segmento E

4. Segmento C
5. Ânodo comum (+5Vcc) do dígito à direita
6. Segmento B
7. Segmento A
8. Segmento F
9. Segmento G
10. Ânodo comum (+5Vcc) do dígito à esquerda

Para acionar o display, usaremos uma biblioteca já existente que facilitará muito a nossa vida. Ela chama-se `SevSeg.h`, de *Seven Segments* (Sete Segmentos), e pode ser obtida em https://github.com/fbryan/sevseg. Use o que foi explicado no projeto nº 02 (*capítulo 3*) para instalá-la.

Para essa biblioteca, teremos de passar os pinos onde estão ligados os segmentos de LED do display de 7-segmentos e onde estarão ligados os pinos comuns do display. Podemos usar tanto os pinos digitais quanto analógicos para essa tarefa — o que é excelente, pois não nos restam muitos pinos digitais para ligar o display e, se dependêssemos apenas deles, não teríamos o suficiente.

Para ligar o display ao Arduino, use o seguinte esquema:

1. Coloque um resistor de, pelo menos, 330Ω em cada terminal ânodo comum, ou seja, nos pinos 5 e 10, conforme a figura:

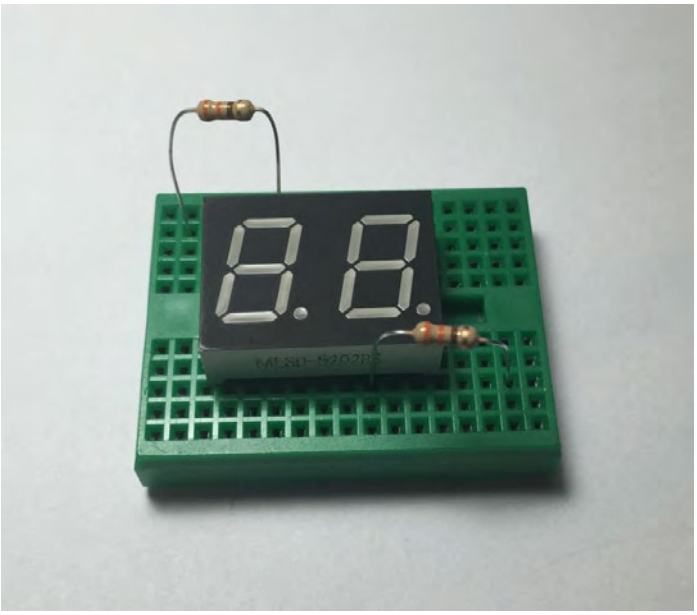


Figura 4.12: Resistores nos ânodos do display

2. Agora ligue os resistores dos terminais comuns ao Arduino, sendo que o do dígito à direita (5) deve ir no pino analógico A0, e o do dígito à esquerda no pino analógico A1.

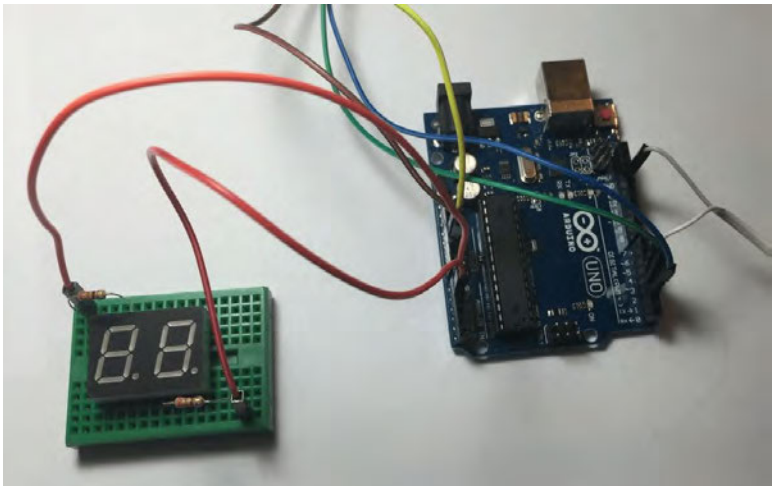


Figura 4.13: Ânodos comuns ligados ao Arduino

3. Ligue:

    - O terminal do segmento A (7) ao pino digital 5 do Arduino;
    - O terminal do segmento B (6) ao pino digital 6 do Arduino;
    - Terminal do segmento C (4) ao pino digital 7;
    - Terminal do segmento D (1) ao pino digital 8;
    - Terminal do segmento E (3) ao pino digital 9;
    - Terminal do segmento F (8) ao pino digital 10;
    - Terminal do segmento G (9) ao pino digital 11;
    - Terminal do ponto decimal (2) ao pino digital 12.

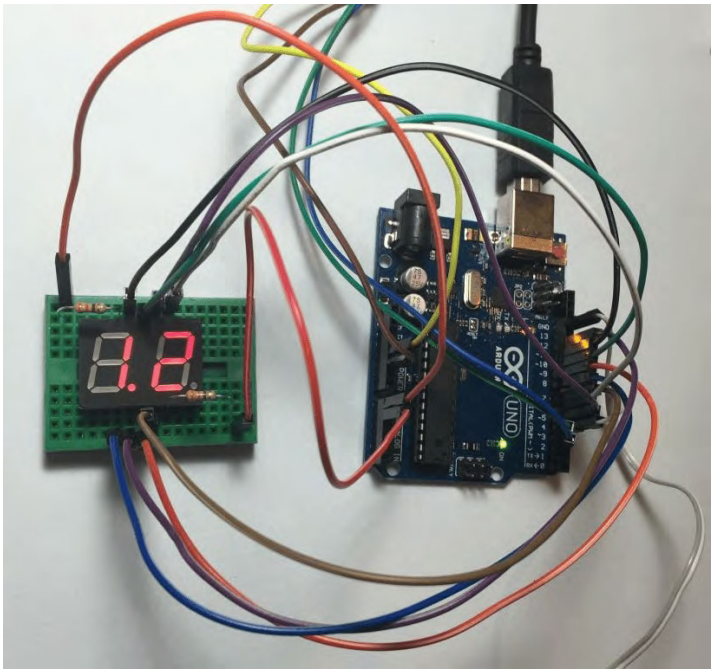


Figura 4.14: Todos os terminais do display ligados ao Arduino

Vamos testar para ver se está funcionando corretamente. Use o código seguinte para mostrar o número `1.2` no display de 7-segmentos:

```
#include <SevSeg.h>

SevSeg display;

void setup() {
  byte digitos = 2;
  byte pino_digitos[] = {A0, A1};
  byte pino_segmentos[] = {5, 6, 7, 8, 9, 10, 11, 12};
  display.begin(COMMON_ANODE, digitos, pino_digitos, pino_segmento
s);
}

void loop() {
  display.setNumber(12,1);
  display.refreshDisplay();
}
```

Depois de compilar e enviar ao seu Arduino, você deverá ver o número `1.2` (o número 1 no dígito à esquerda, primeiro ponto decimal aceso, o número 2 no dígito à direita e segundo ponto decimal apagado), como na figura anterior ao código.

Para esse código, incluímos a biblioteca `SevSeg.h` com a linha `#include<SevSeg.h>` e, já em seguida, declaramos o objeto `display` como sendo do tipo `SevSeg` com a linha `SevSeg display`. Na função `void setup()`, declaramos a variável `digitos` como sendo do tipo `byte` e inicializada com o número 2. Ela determinará a quantidade de dígitos existentes em nosso display de 7-segmentos.

O vetor do typo `byte`, chamado `pino_digitos[]`, armazena os pinos onde estão ligados os ânodos (ou cátodos, se fosse o caso) dos dígitos do display. O vetor também do tipo `byte`, chamado `pino_segmentos[]`, armazena onde estão ligados os segmentos de LED do display, na sequência alfabética dos segmentos e, por último, o ponto decimal.

Com o método `display.begin(COMMON_ANODE, digitos, pino_digitos, pino_segmentos)`, inicializamos o display indicando `COMMON_ANODE`, porque ele é ânodo comum. Se fosse cátodo comum, usaríamos `COMMON_CATHODE`. Em seguida, indicamos a quantidade de dígitos do display, onde estão os pinos comuns e os pinos dos segmentos.

Na função `loop()`, apenas usamos o método `setNumber` como `display.setNumber(12,1)`, indicando que os números a serem mostrados são o 1 no dígito à esquerda, 2 no dígito à direita, e o primeiro ponto decimal aceso (1). Para acender o segundo ponto decimal, devemos usar 0 em vez de 1.

Lembre-se apenas de que o display só pode mostrar no máximo a quantidade de dígitos que possuir. O método

`display.refreshDisplay()` faz com que o número seja efetivamente mostrado.

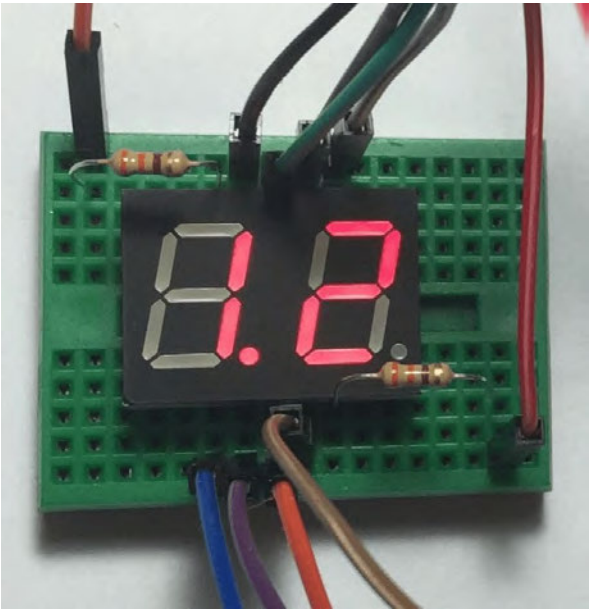


Figura 4.15: 1.2 no display de 7-segmentos

Agora vamos aproveitar e fazer mais um teste com um código para executar o contador de zero até 99. Assim, já poderemos vislumbrar como juntar tudo.

```
#include <SevSeg.h>
SevSeg display;

void setup() {
  byte digitos = 2;
  byte pino_digitos[] = {A0, A1};
  byte pino_segmentos[] = {5, 6, 7, 8, 9, 10, 11, 12};
  display.begin(COMMON_ANODE, digitos, pino_digitos, pino_segmento
s);
}

void loop() {
  for(int i=0;i<=99;i++) {
    display.setNumber(i,0);
    display.refreshDisplay();
```

```
    delay(10);
  }
}
```

A configuração ficou a mesma, portanto, só alterei a função `loop()` colocando uma estrutura de repetição `for` para contar de 0 até 99 e incrementos de 1: `for(int i=0;i<=99;i++)`.

Em seguida, alterei a linha do método `setNumber` para usar a variável `i`, que é a contadora do `for`: `display.setNumber(i,0)`. Depois é só fazer com que os números sejam mostrados com `display.refreshDisplay()` e aguardar um pequeno tempo (10 milissegundos), para que seja possível vê-los mudar com `delay(10)`.

Agora que já compreendemos como tudo funciona, ficou fácil. Vamos juntar tudo e terminar nosso radar!

Vamos usar os códigos do sensor ultrassônico que já fizemos anteriormente com o do display de 7-segmentos que acabamos de fazer:

```
#include <Ultrasonic.h>
#include <SevSeg.h>

Ultrasonic sensor(3, 2);
SevSeg display;

int botao = 4;

float anterior = 0;
float distancia = 0;
float deslocamento = 0;

void setup()
{
  pinMode(botao, INPUT_PULLUP);
  byte digitos = 2;
  byte pino_digitos[] = {A0, A1};
  byte pino_segmentos[] = {5, 6, 7, 8, 9, 10, 11, 12};
  display.begin(COMMON_ANODE, digitos, pino_digitos, pino_segmento
s);
}
```

```
void loop()
{
  if (digitalRead(botao) == LOW) {
    long tempo = sensor.timing();
    float medida = round(sensor.convert(tempo, Ultrasonic::CM));
    distancia = abs(medida - anterior);
    deslocamento = (distancia * 10) / 100;
    anterior = medida;
  }
  display.setNumber(deslocamento,1);
  display.refreshDisplay();
}
```

Depois de compilar e enviar ao Arduino, você deve apertar o botão e movimentar alguma coisa em frente ao sensor ultrassônico para ver a velocidade no display de 7-segmentos. Ao soltar o botão, o número da última medição deverá permanecer no mostrador.

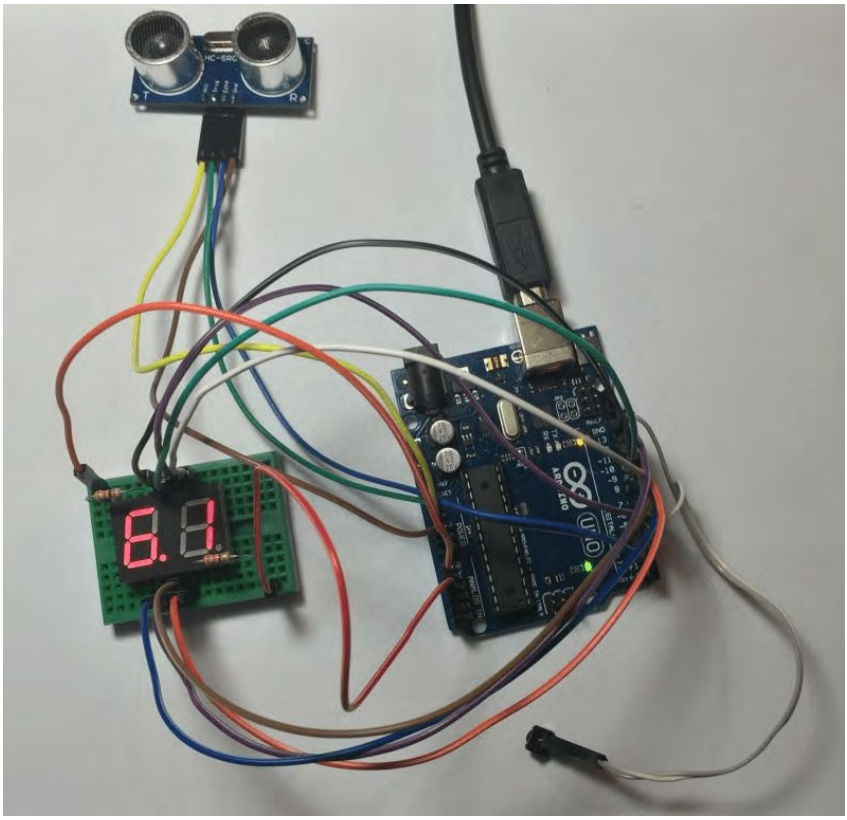


Figura 4.16: Tudo montado e funcionando

Você pode ver um vídeo desse projeto em funcionamento em https://youtu.be/wOZspVR4sT8.

Tudo isso se tornou irresistível de se colocar em uma pistola de videogame, e deixar a criançada brincar como sugeri na introdução:

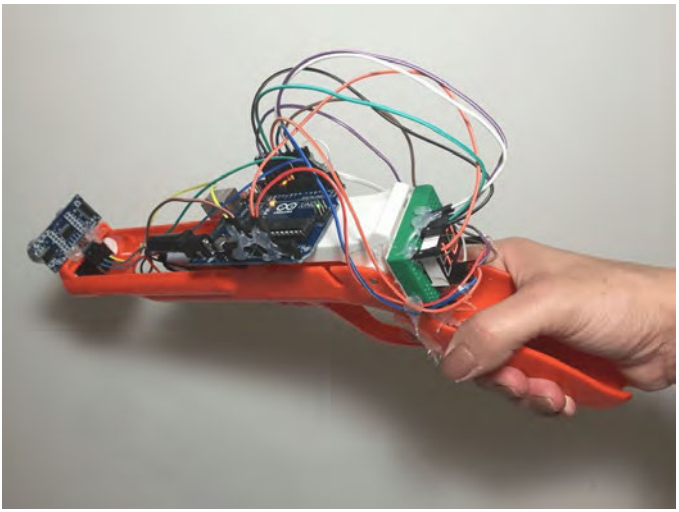


Figura 4.17: O projeto montado em uma pistola de videogame

4.3 DESAFIO

Tente colocar todo o projeto dentro de uma pistola, como fiz e mostrei anteriormente, para imitar os radares policiais que vemos nas estradas brasileiras! Você pode construir sua própria pistola se não tiver uma que serve para encaixar controles remotos de videogame. Assim, seu projeto ganha mobilidade, e com certeza poderá ser útil para explicar conceitos de física, no caso propagação e velocidade do som, em feiras de ciência e aulas.

Depois de tanto movimento nesse projeto, o próximo é para quem gosta de comodidade e não quer se movimentar tanto assim: que tal acionar as lâmpadas da sua casa com um controle remoto?

CAPÍTULO 5

PROJETO Nº 04 — QUE TAL ACIONAR AS LÂMPADAS DE SUA CASA COM CONTROLE REMOTO?

E a preguiça de levantar para apagar a luz na hora de dormir, ou assistir um filme no escurinho da sala ou do quarto? Com esse projeto, você poderá controlar uma lâmpada, seja ela a do teto ou de um abajur, apenas usando um controle remoto infravermelho de forma bem simples.

É possível, inclusive, criar algum código secreto que apenas acionará a iluminação caso seja repetido pelo usuário. Assim você poderá acionar lâmpadas diferentes em um mesmo ambiente.

5.1 MATERIAIS UTILIZADOS NESSE PROJETO

- 1 x Arduino UNO R3
- 1 x Prot-o-board
- 1 x Sensor infravermelho
- 1 x Controle remoto
- 1 x Resistor de 1 KΩ 1/4W

- 1 x Transistor TIP102
- 1 x Relê 5V/125V
- 1 x Abajur (110V com lâmpada de 7W)
- 1 x Ferro de solda
- Estanho para solda
- Fios diversos

5.2 DESENVOLVENDO O PROJETO

O circuito é basicamente o mesmo do projeto nº 02 (*capítulo 3*), mas usaremos um controle remoto para controlar tudo a distância. Para começar, vamos precisar da dupla: receptor infravermelho e controle remoto.

Figura 5.1: O controle remoto e o receptor infravermelho

Esses receptores normalmente já vêm acompanhados do controle remoto e podem ser encontrados em lojas especializadas em eletrônica e em Arduino. Mas se você quiser usar qualquer outro controle remoto, também é possível. Portanto, se assim desejar, basta obter apenas o receptor de infravermelho. Imagine, você pode usar o controle da televisão e, com a sequência correta de teclas, controlar qualquer luz!

A montagem é simples. O receptor tem um pino marcado com uma letra `S`, e este é o pino de sinal. Por ele, receberemos os valores das teclas pressionadas. O pino do meio é o +5V e o da esquerda é o GND.

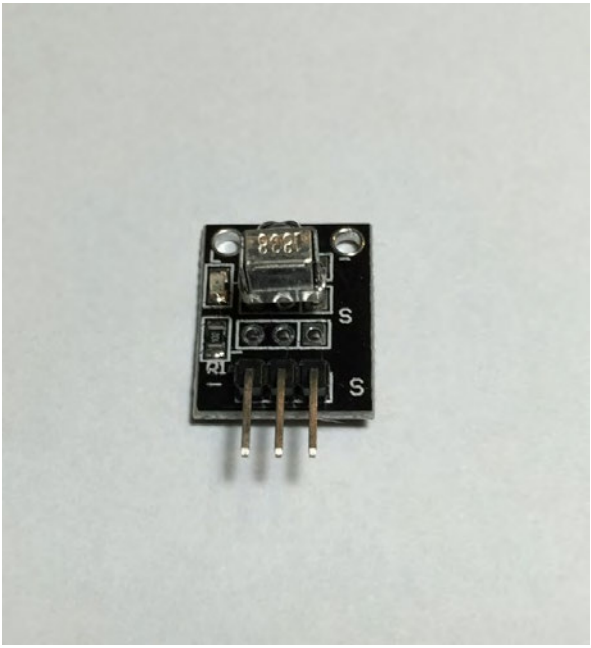


Figura 5.2: O receptor infravermelho com o pino de sinal marcado com a letra "S"

A montagem é conectar o pino de sinal do receptor ao pino digital 3 do Arduino; o pino +5V do receptor ao pino 5V do Arduino; e o pino GND do receptor a um dos pinos GND do

Arduino.

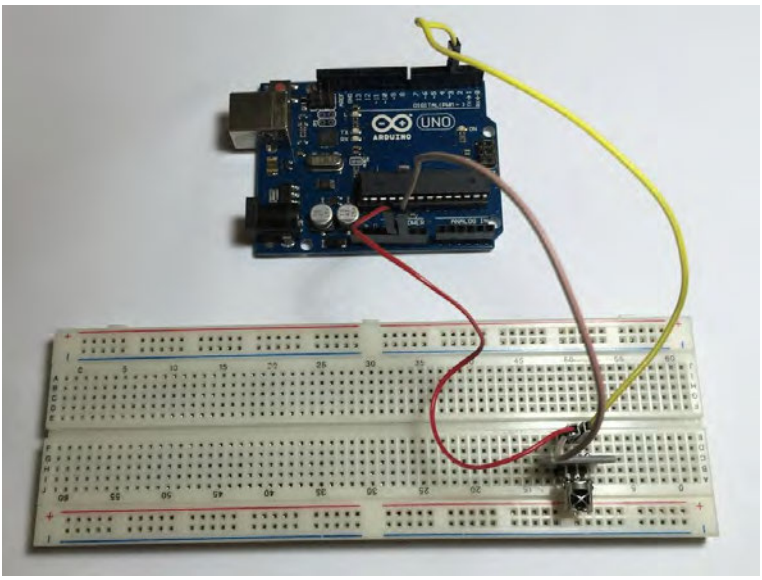


Figura 5.3: A ligação entre o receptor infravermelho e o Arduino

Para podermos capturar as teclas do controle remoto através do receptor infravermelho, teremos de usar uma biblioteca IRLib, que pode se obtida em https://github.com/fbryan/irlib. Basta fazer download do arquivo ZIP e, para incluí-la ao IDE Arduino, faça do mesmo jeito que já foi apresentado também no projeto nº 02 (*capítulo 3*).

Com a biblioteca já inclusa no IDE, vamos primeiro testar a recepção das teclas do controle remoto que você queira usar. Cada tecla envia ao receptor um valor diferente, que precisa ser lido e interpretado para que realizem tarefas conforme o seu programa determinar.

Vamos usar o pequeno programa a seguir para obter esses valores e anotá-los, para podermos montar tudo e fazer o programa que controlará a luz.

```
#include <IRLib.h>

IRdecode decodificador;
IRrecv receptor(3);

void setup() {
  Serial.begin(9600);
  receptor.enableIRIn();
  delay(100);
}

void loop() {
  if(receptor.GetResults(&decodificador)) {
    decodificador.decode();
    delay(100);
    Serial.println(decodificador.value, HEX);
    receptor.resume();
  }
}
```

Carregue-o para o Arduino e abra o Serial Monitor. Aponte o controle para o receptor e comece a pressionar as teclas. Você verá os valores correspondentes a cada uma delas depois que forem decodificados.

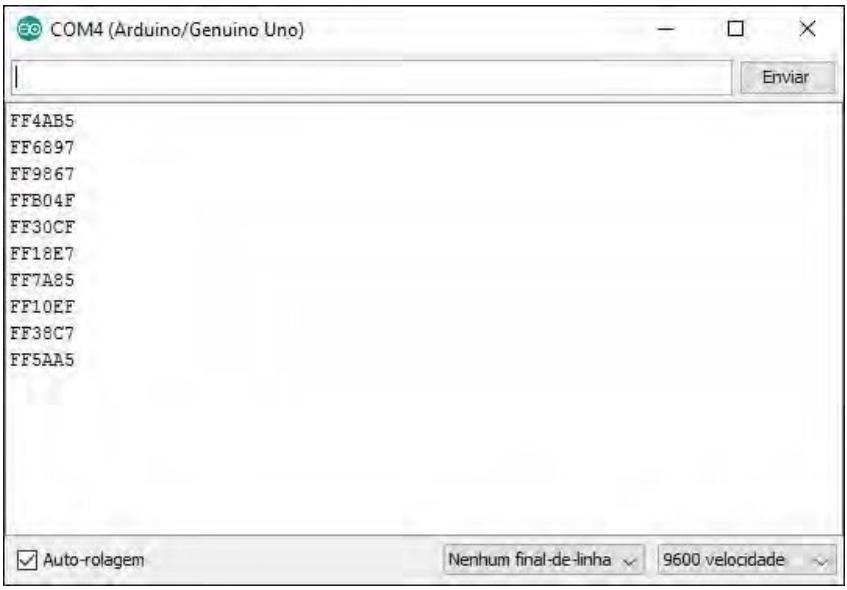


Figura 5.4: Valores das teclas de 0 até 9 do controle remoto que acompanhou o receptor

Com esse programa, podemos saber o valor das teclas de qualquer controle remoto. Você pode usá-lo para testar com o controle da TV, do receptor de TV por assinatura, do aparelho de som, enfim, de qualquer controle mesmo. Basta ir testando-os.

Começamos por importar a biblioteca que baixamos e incorporamos ao IDE Arduino, a `IRLib.h` . Ela nos dará uma série de ferramentas que tornará o trabalho mais fácil de receber e decodificar o sinal que estamos recebendo.

Com `IRdecode decodificador` , estamos instanciando um objeto chamado `decodificador` do tipo `IRdecode` , que nos permite decodificar e interpretar vários aspectos do sinal recebido, como a quantidade de bits, valor e protocolo.

Já com `IRrecv receptor(3)` , instanciamos um objeto chamado receptor que passa como parâmetro o pino digital por onde serão recebidos os sinais do receptor. Esse objeto é do tipo `IRrecv` que, como o `IRdecode` , nos permite obter uma série de controles sobre o receptor, como inicializá-lo ou reinicializá-lo.

Na função `setup()` , inicializamos uma comunicação serial com `Serial.begin(9600)` para podermos depois ver os valores recebidos pelo Serial Monitor. Usamos `receptor.enableIRIn()` para inicializar o receptor de infravermelho, e damos a ele um tempo de 100 milissegundos para iniciar com `delay(100)` .

Na função `loop()` , usamos uma estrutura de seleção `if (receptor.GetResults(&decodificador))` para verificar se há um sinal recebido. Caso exista, o Arduino tenta decodificá-lo; caso contrário, ele não deverá fazer nada. Ou seja, apenas se houver um sinal a ser decodificado tentaremos decodificá-lo.

A linha `decodificador.decode()` faz com que o sinal seja decodificado e seus dados guardados em uma estrutura dentro do

objeto decodificado, sendo que alguns dos campos que podem ser acessados são:

Campo	Tipo	Descrição
decode_type	char	Nome do protocolo do sinal decodificado (NEC, SONY, RC5 etc.)
value	long	Valor decodificado
bits	char	Quantidade de bits no valor decodificado

É preciso dar um tempo para que o Arduino possa processar a decodificação. Então, usamos `delay(100)` para dar-lhe 100 milissegundos para isso, o que é suficiente.

Na linha `Serial.println(decodificador.value, HEX)`, mostramos o campo `value` mencionado anteriormente no Serial Monitor no formato hexadecimal. Ou seja, mostramos o valor do sinal correspondente a uma tecla do controle remoto que foi recebido. Para encerrar o ciclo, antes de começar tudo de novo, reiniciamos o receptor com `receptor.resume()`, para que ele possa fazer a próxima leitura.

Bom, já temos os valores que desejamos, agora vamos ao programa que controlará a lâmpada. Usarei a mesma estratégia do projeto nº 02 (*capítulo 3*): primeiro faremos com que o LED do pino digital 13 acenda quando a tecla correta for pressionada, depois faremos o circuito para o relê com a lâmpada propriamente dita.

```
#include <IRLib.h>

IRdecode decodificador;
IRrecv receptor(3);

int led = 13;
long valor = 0;

void setup() {
  pinMode(led,OUTPUT);
  receptor.enableIRIn();
```

```
  delay(100);
}

void loop() {
  if(receptor.GetResults(&decodificador)) {
    decodificador.decode();
    delay(100);
    valor = decodificador.value;
    if(valor == 0xFF4AB5) {
      digitalWrite(led,HIGH);
      delay(1000);
      digitalWrite(led,LOW);
    }
    receptor.resume();
  }
}
```

Esse programa é apenas uma mudança do programa anterior, poucas mudanças como você pode verificar. Apenas declaramos variáveis para identificar o pino digital onde o LED está conectado, e outra para receber o valor que for lido pelo receptor infravermelho.

Na função `setup()`, setamos o pino digital do LED como saída, e iniciamos o receptor infravermelho, dando-lhe 100 milissegundos para completar a tarefa. Na função `loop()`, caso tenhamos algum sinal para ser decodificado, fazemos isso e guardamos o valor lido na variável `valor`, com a linha `valor = decodificador.value`.

Com uma estrutura de seleção `if(valor == 0xFF4AB5)`, podemos verificar se a tecla zero do controle remoto foi pressionada. Usamos esse valor, pois ele foi lido com o programa anterior em que testamos as nossas teclas de interesse do controle remoto. Para o seu controle, pode ser um valor diferente, por isso a importância de usar o programa de teste antes.

Bom, quando a tecla que desejamos é pressionada, acendemos o LED por um segundo e depois o deixamos apagado. O que, convenhamos, não faz nenhum sentido: ninguém vai querer

acender a luz usando um controle remoto para ela ficar acesa apenas um segundo! Vamos melhorar muito o nosso programa de forma bem simples.

```
#include <IRLib.h>

IRdecode decodificador;
IRrecv receptor(3);

int led = 13;
long valor = 0;

void setup() {
  pinMode(led,OUTPUT);
  receptor.enableIRIn();
  delay(100);
}

void loop() {
  if(receptor.GetResults(&decodificador)) {
    decodificador.decode();
    delay(100);
    valor = decodificador.value;
    if(valor == 0xFF4AB5) {
      digitalWrite(led,!digitalRead(led));
      delay(1000);
    }
    receptor.resume();
  }
}
```

Note que apenas trocamos as linhas que estavam dentro da estrutura de seleção `if(valor == 0xFF4AB5)` , na qual verificamos se o valor lido é o da tecla zero do controle remoto. Removemos também as linhas:

```
digitalWrite(led,HIGH);
delay(1000);
digitalWrite(led,LOW);
```

Elas acendiam o LED do pino digital 13 por apenas um segundo. Assim, acrescentamos as linhas:

```
digitalWrite(led,!digitalRead(led));
delay(1000);
```

Nelas, lemos o estado do pino digital 13 (`HIGH` ou `LOW`), e aplicamos seu inverso nele mesmo. Para isso, usamos `!digitalRead(led)` , em que o sinal de exclamação significa o operador lógico `NOT` , ou seja, a inversão do sinal que, se estiver em `HIGH` , será invertido para `LOW` e vice-versa.

Ainda assim, damos um segundo de espera para caso você mantenha a tecla apertada, o que causaria o acendimento e desligamento repetidas vezes da luz, de forma muito rápida, podendo danificar o relê ou até queimar a lâmpada. Com isso, o tempo mínimo entre ligar e desligar, ou entre desligar e ligar, será de um segundo.

Vamos então à parte mais crítica: ligar uma lâmpada de verdade e controlá-la com o Arduino através de um relê.

> Antes de começar, lembre-se de tomar todo cuidado necessário quando se utiliza a eletricidade. Siga todas as normas de segurança tal como realizar a montagem com todos os equipamentos desligados, proteger e isolar os contatos, e não tocar partes metálicas e contatos enquanto o equipamento estiver em funcionamento.
>
> Os riscos são sérios e você deve realmente ter cuidado para não sofrer um acidente. Em caso de dúvida, reveja, reestude e procure ajuda.

Como esse projeto funciona do mesmo jeito que o projeto nº 02 (*capítulo 3*), lá você já tem todas as explicações do funcionamento e do circuito para ligar o relê. Portanto, vou pular essa parte. Você pode voltar a relê-lo caso não se lembre de algum detalhe.

Primeiro, vamos a ligação do transistor TIP102 na prot-o-board:

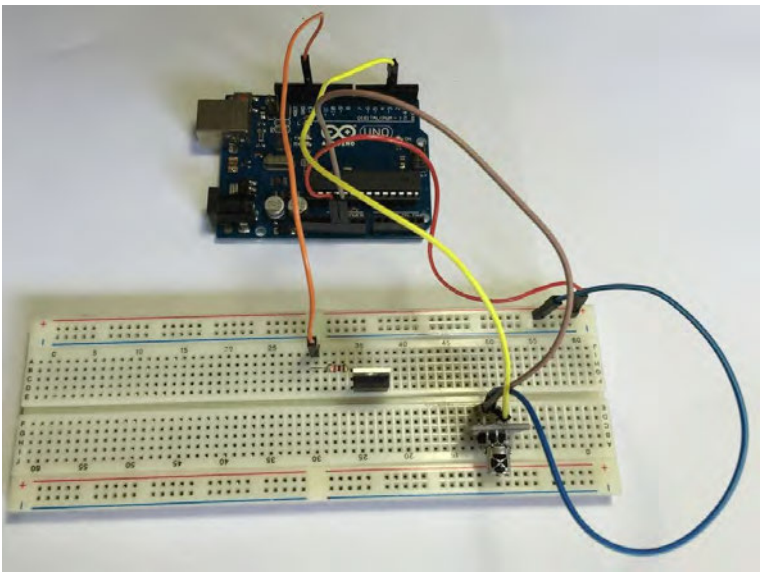


Figura 5.5: O transistor TIP102 ligado na prot-o-board e conectado ao Arduino

Conecte o terminal base do transistor a um resistor de 1KΩ, e o outro terminal do resistor ao pino digital 13 do Arduino. Assim, não mudaremos absolutamente nada no programa. Note na figura anterior que mudei a alimentação dos 5V do sensor de infravermelho para facilitar a alimentação também do relê.

Ao terminal emissor do transistor, ligue um dos pinos GND do Arduino. O terminal coletor do transistor deve ser ligado a um dos terminais da bobina do relê. E o outro terminal da bobina do relê deve ser ligado ao 5V, que pode ser usado na prot-o-board em conjunto com a alimentação do sensor infravermelho.

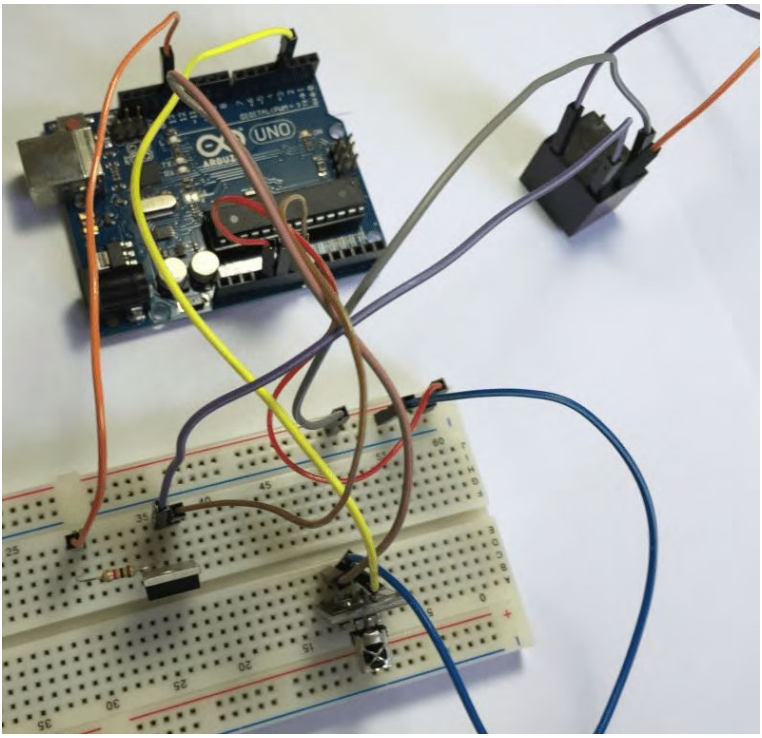


Figura 5.6: O transistor ligado ao GND do Arduino e à bobina do relê, e o relê ligado ao 5V do Arduino

Entre os contatos do relê para o chaveamento normalmente aberto, vamos colocar nossa lâmpada. No meu caso, vou usar um abajur, que é um velho companheiro e que, a partir de agora, receberá um upgrade! Vou colocar o relê em paralelo com o botão liga/desliga dele.

Figura 5.7: O abajur

Com os devidos cuidados, os fios que vão no relê serão ligados aos contatos do botão:

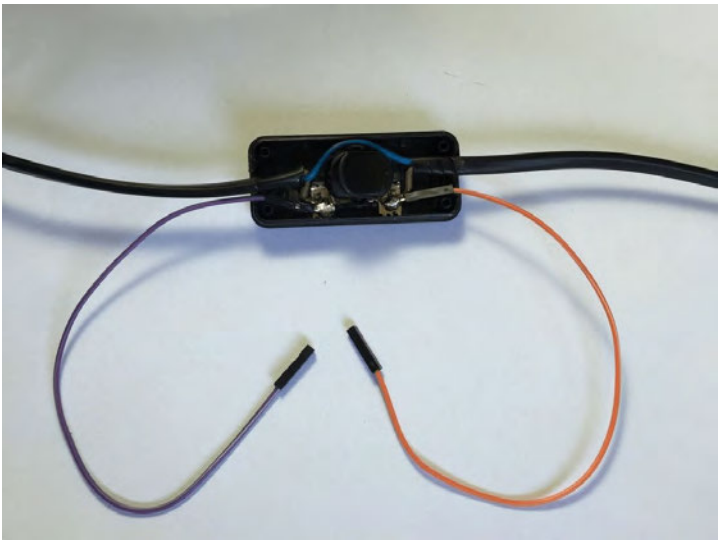


Figura 5.8: Os fios que serão ligados aos contatos normalmente abertos do relê

Note que estou usando fios finos e comuns, porque a lâmpada que está no abajur é de baixa potência (7W), então a corrente que fluirá pelos fios é baixíssima (aproximadamente 0,05A). É extremamente recomendável que você use fios de bitolas adequadas para a quantidade de corrente que será usada para o tipo de lâmpada que estiver utilizando no seu projeto.

A ligação entre o interruptor e o relê é bem simples, bastando ligar os fios aos terminais do contato normalmente aberto do relê.

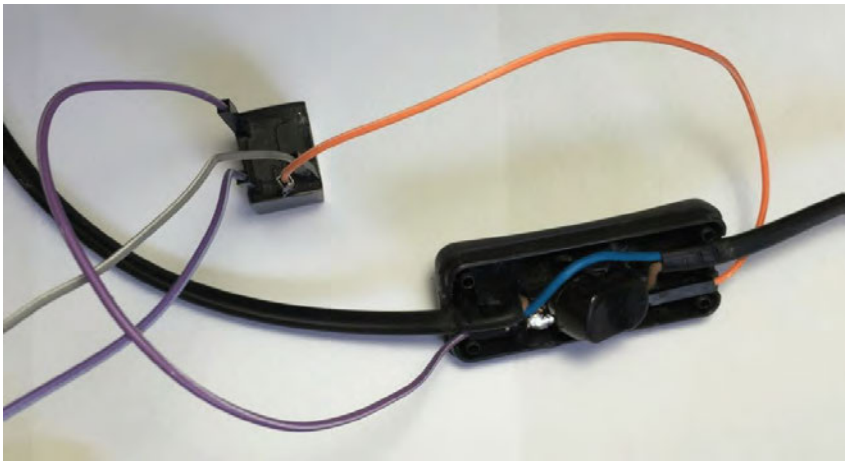


Figura 5.9: Os fios do interruptor aos terminais do contato normalmente aberto do relê

Depois de tudo ligado, você pode testar o funcionamento geral. Agora sim, coloque o abajur na tomada e ligue seu Arduino ao computador. Cuidado para não encostar em algum contato que resulte em choque elétrico.

Aponte o controle remoto diretamente para o sensor infravermelho e pressione a tecla zero. A lâmpada do abajur deverá acender. Aguarde um segundo e pressione novamente a tecla zero, para a lâmpada do abajur apagar.

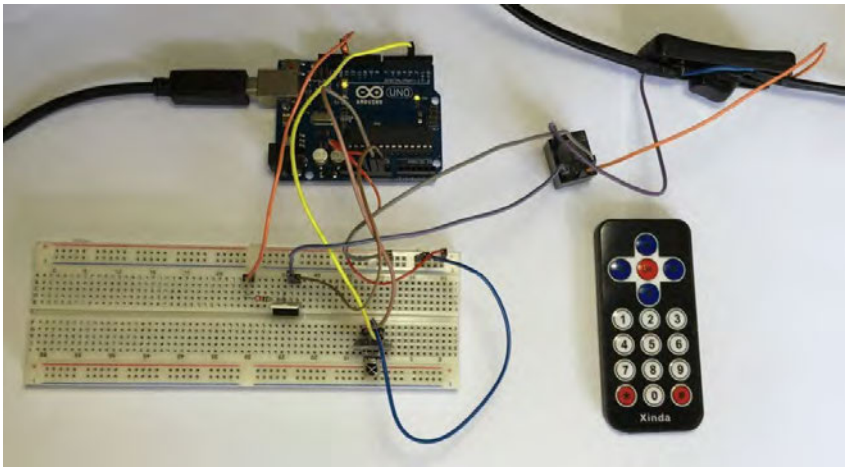


Figura 5.10: Visão geral do projeto montado

Agora você pode usar esse mesmo esquema de montagem para automatizar qualquer lâmpada de sua casa, desde abajures até a iluminação geral.

Você encontrará um vídeo desse projeto em funcionamento em https://youtu.be/MWAEu2303jM.

5.3 DESAFIO

O primeiro desafio de todos é fazer esse circuito ficar pequeno o bastante para ser embutido discretamente nos abajures ou nos interruptores da iluminação da casa. Se não, dentro dos interruptores, discretamente em uma caixa ao lado ou próximo deles.

Para isso, você pode usar o projeto nº 01 (*capítulo 2*) e começar construindo seu próprio Arduino. Depois pode ir adicionando os outros componentes e, claro, criar uma PCB (*Printed Circuit Board*

— placa de circuitos impressos) específica para tudo isso — que é sempre uma excelente ideia.

Outro desafio que pode ser testado ainda na prot-o-board é criar combinações de teclas para acionar a iluminação. Por exemplo, você usa o próprio controle remoto da TV e, a partir de uma combinação que não represente um canal, você aciona a luz.

Há ainda a possibilidade de acionar várias lâmpadas a partir de um único Arduino. Nesse caso, basta replicar o conjunto transistor/relê e, com apenas um sensor, você pode acionar a quantidade de lâmpadas correspondente à quantidade de pinos digitais que estiverem livres.

Solte a imaginação. Desafie-se e boa diversão!

Gosta de jogar jogos de tabuleiro? Que tal retomar esses jogos, mas com um novo atrativo: um dado eletrônico? No próximo projeto, vamos desenvolver um dado usando uma matriz de LEDs. Tente adivinhar qual será o próximo número sorteado, e vamos lá!

CAPÍTULO 6

PROJETO Nº 05 — UM DADO ELETRÔNICO PARA JOGOS

Da próxima vez que for jogar um jogo de tabuleiro, ou qualquer outra atividade que dependa do sorteio de dados, leve o seu superdado eletrônico feito com Arduino e LEDs!

Este é um dos projetos mais divertidos de construir e que gera maior repercussão entre a "plateia". Nós nos reunimos para jogar Banco Imobiliário, War etc. e, quando todos veem o dado funcionando, parece que gera uma excitação extra ao jogo! Sugestões não faltam.

6.1 MATERIAIS UTILIZADOS NESSE PROJETO

- 1 x Arduino
- 9 x LEDs vermelhos
- 3 x Resistores de 220Ω
- 1 x Ferro de solda
- Estanho para solda
- Fios diversos

6.2 DESENVOLVENDO O PROJETO

A ideia é imitar um dado mesmo, mostrando os números (bolinhas) de 1 até 6 e sorteando os valores aleatoriamente. Mas antes de qualquer coisa, é preciso construir uma matriz de LEDs de três colunas por três linhas, ou seja, 9 LEDs.

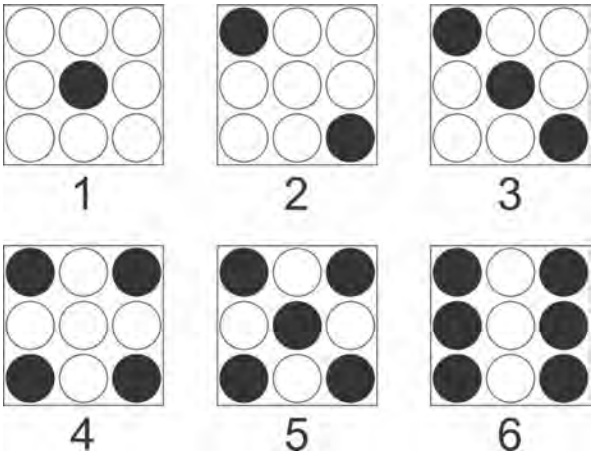


Figura 6.1: Faces possíveis de um dado comum de quatro lados. Note a necessidade de uma matriz de 3x3

Vamos então pelo começo: o LED. Com um LED em mãos, veja que ele tem um terminal mais curto do que o outro. Esse terminal mais curto é o negativo, e o mais longo, o positivo do LED.

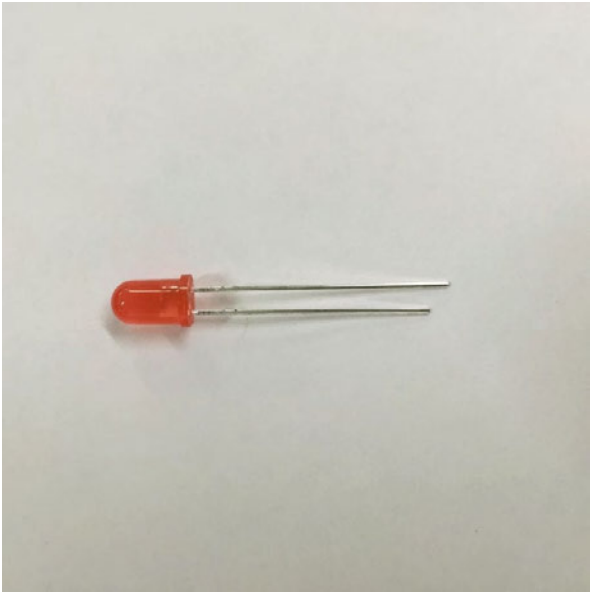


Figura 6.2: Um LED com os terminais de tamanhos diferentes

Você terá de se atentar em ligar os terminais negativos sem misturá-los com os positivos, portanto, atenção a esse detalhe. Precisaremos colocar os terminais do LED um para cada lado, mas esse lado precisa ser igual para todos os LEDs. Para facilitar, entorte o terminal negativo, mais curto, mais perto da "cabeça" do LED:

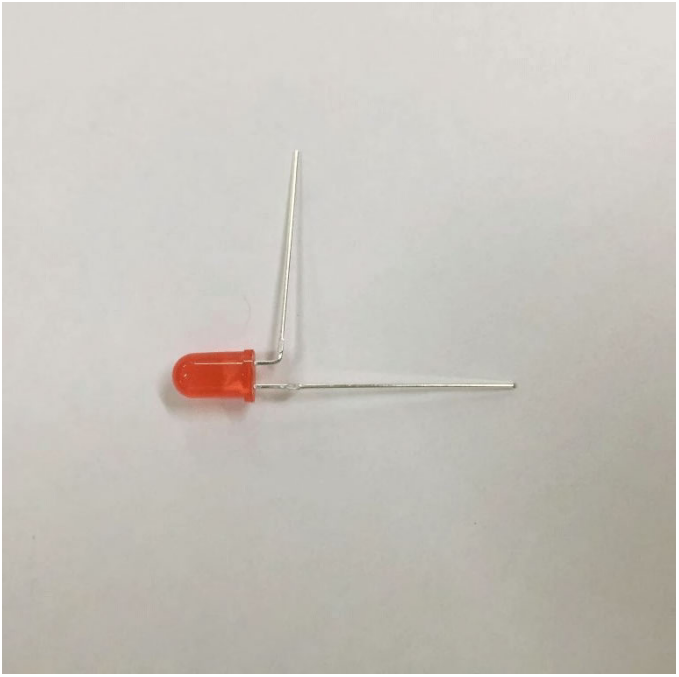


Figura 6.3: Terminal negativo para um lado, com a dobra perto da "cabeça" do LED

Agora dobre o terminal positivo, mais longo, formando um ângulo de 90° graus em relação ao negativo, para deixar visualmente mais fácil de identificar. Faça a dobra mais longe da "cabeça" do LED.

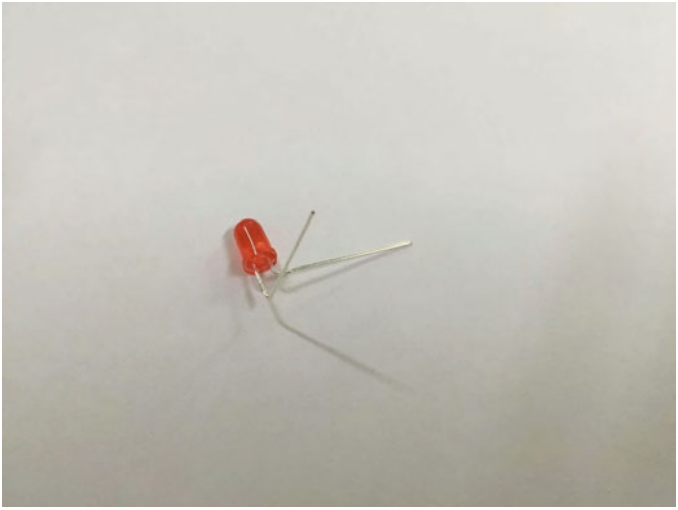


Figura 6.4: Terminal positivo 90° para um outro lado, com a dobra longe da "cabeça" do LED

Veja na imagem a seguir os terminais dobrados, vistos de outro ângulo.

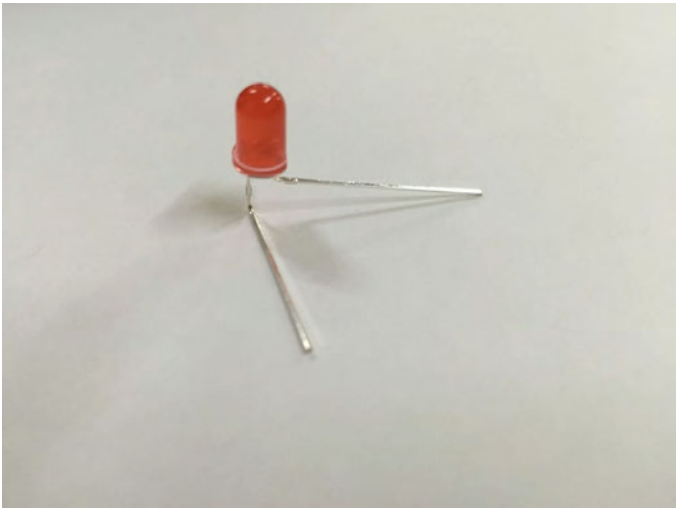


Figura 6.5: Terminais do LED dobrados

Faça a mesma coisa para todos os noves LEDs que formarão

nossa matriz. Depois solde os terminais positivos de três LEDs, como nas figuras a seguir.

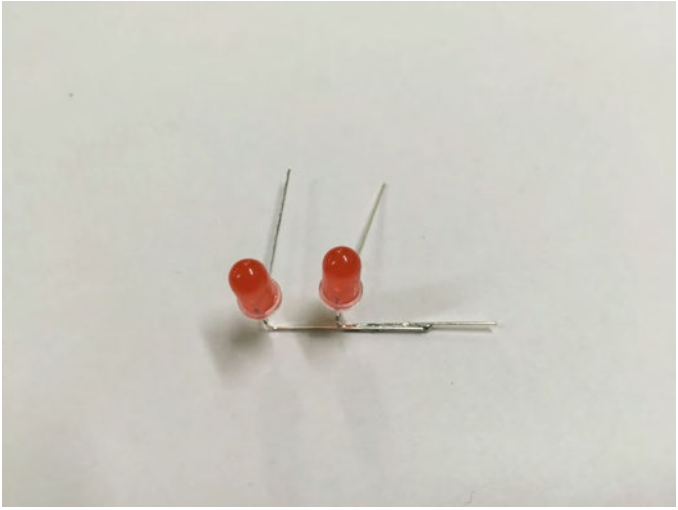


Figura 6.6: Os dois primeiros LEDs com os terminais positivos soldados entre si

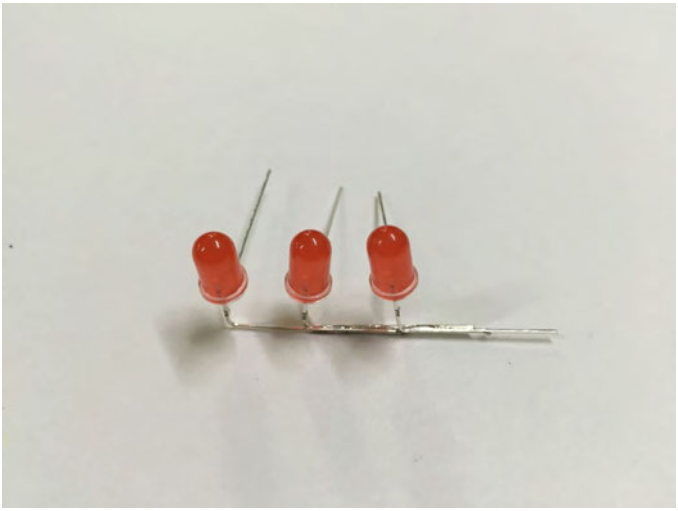


Figura 6.7: Agora o terceiro LED com o terminal positivo soldado aos anteriores

Na seguinte figura, veja os três LED visto por outro ângulo.

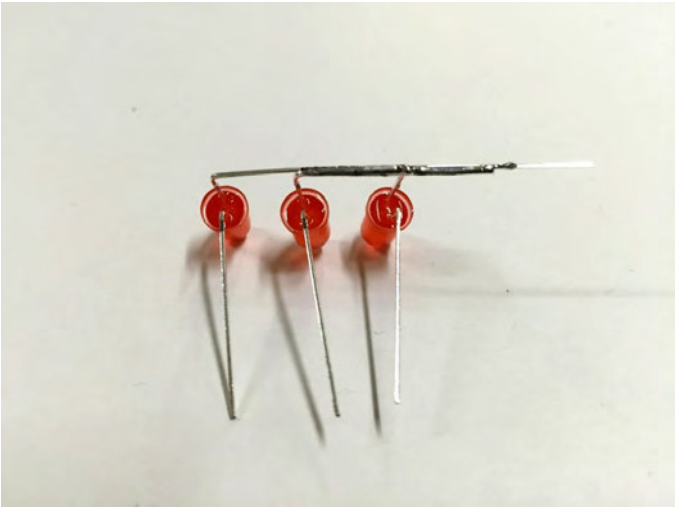


Figura 6.8: Os três LED com os terminais positivos soldados vistos por baixo

Agora teste o funcionamento desse primeiro conjunto: coloque o terminal positivo resultante, que na verdade são os terminais que estão soldados juntos, na prot-o-board. A esse terminal, ligue um resistor de 220Ω e, ao outro terminal do resistor, ligue o pino 5V do Arduino. Depois alterne um fio ligado ao pino GND do Arduino aos terminais negativos dos LEDs.

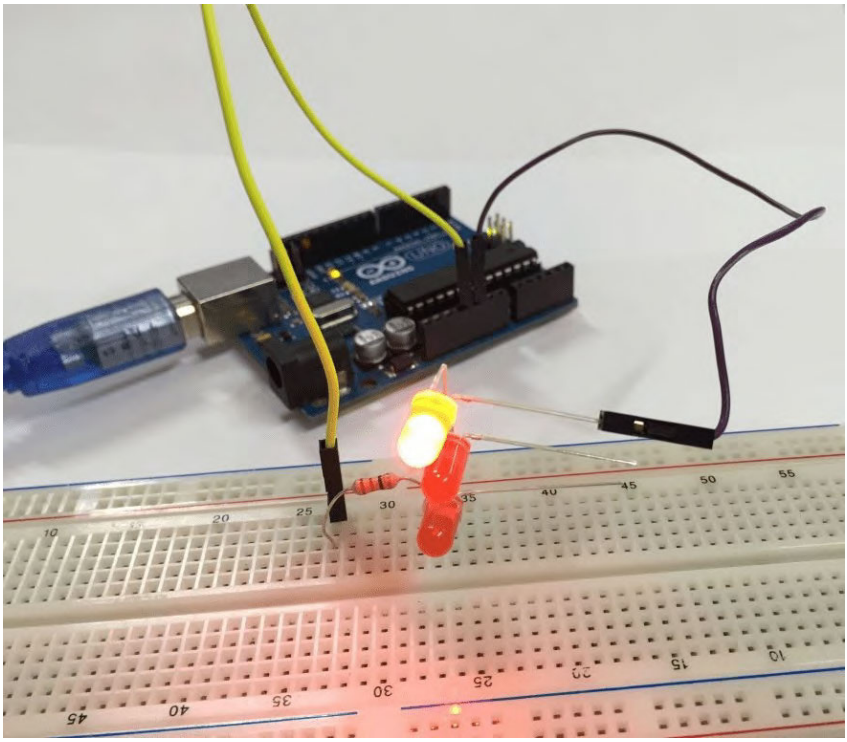


Figura 6.9: Teste do conjunto — Primeiro LED

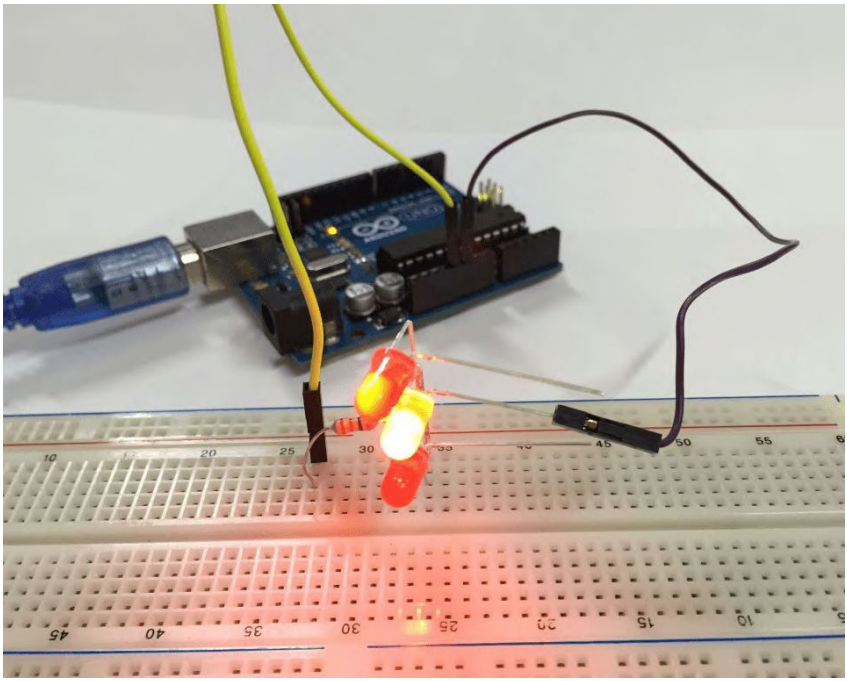


Figura 6.10: Teste do conjunto — Segundo LED

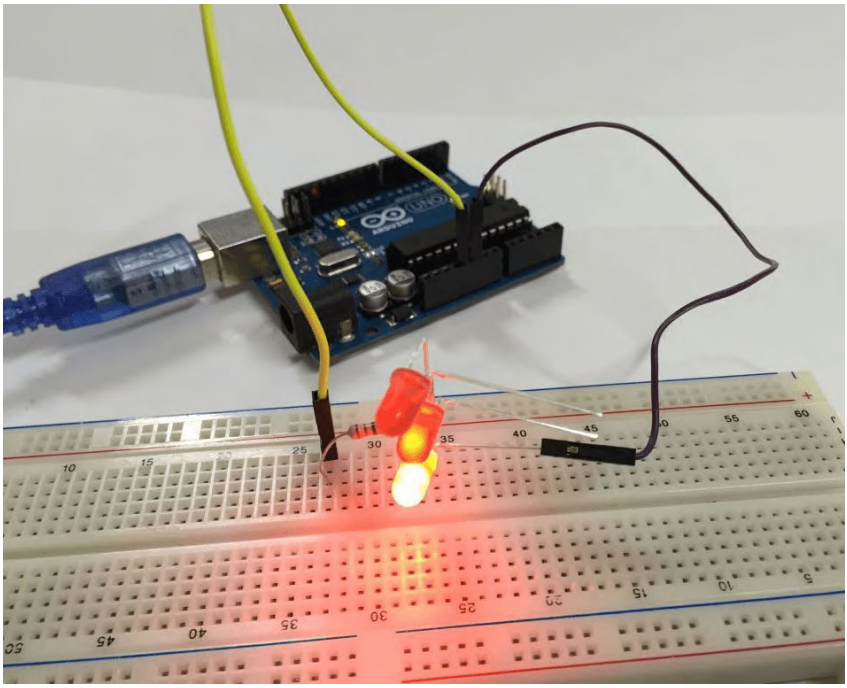


Figura 6.11: Teste do conjunto — Terceiro LED

Com esse primeiro conjunto funcionando, vamos em frente. Faça um segundo conjunto de 3 LEDs e solde os terminais negativos entre os conjuntos, como na figura a seguir:

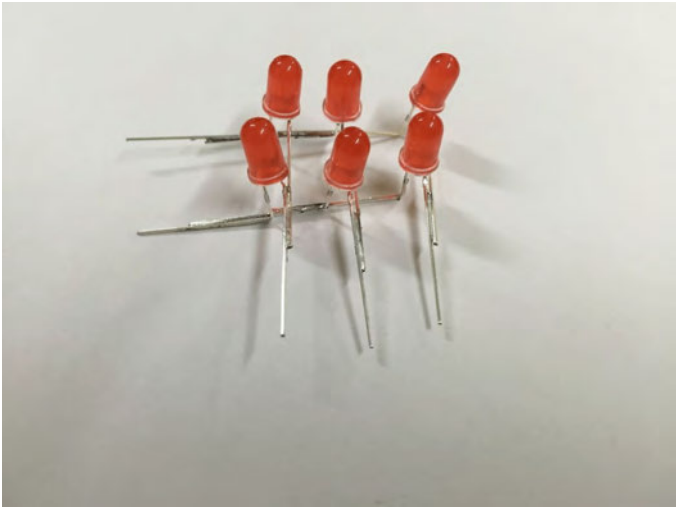


Figura 6.12: Um segundo conjunto de 3 LEDs unidos ao primeiro pelos terminais negativos

Cuidado apenas para que os terminais positivos e negativos não se encostem em qualquer lugar.

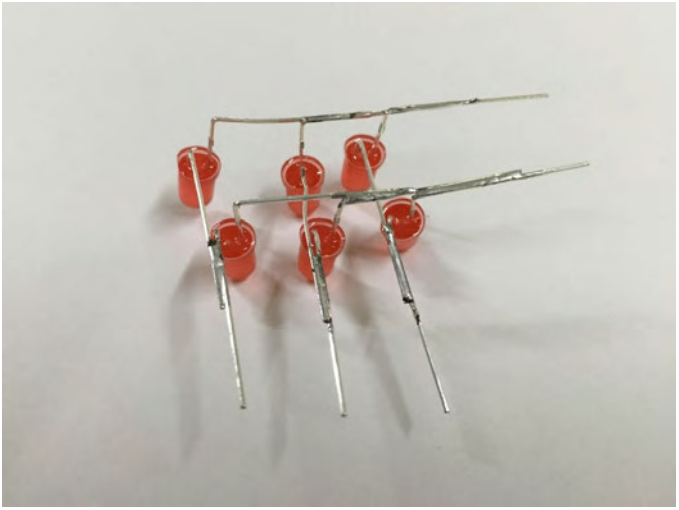


Figura 6.13: O segundo conjunto soldado ao primeiro, visto por baixo

Finalmente, faça o terceiro conjunto de nossa matriz de LEDs, e

solde os terminais negativos desse conjunto aos terminais negativos dos outros dois conjuntos, como na figura:

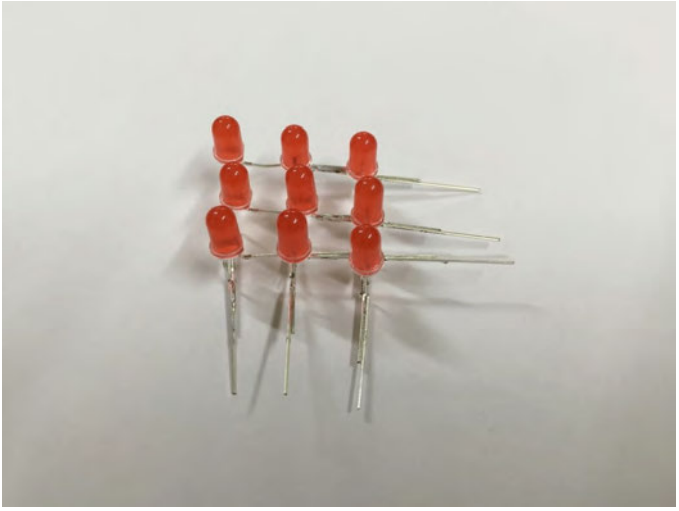


Figura 6.14: Os três conjuntos unidos pelos terminais negativos

Veja agora por outro ângulo:

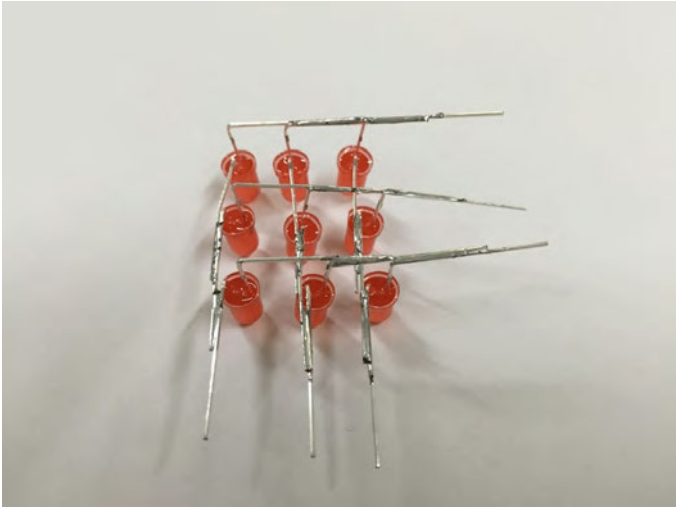


Figura 6.15: Os três conjuntos unidos pelos terminais negativos, vistos por baixo

Finalmente, aos terminais positivos, solde resistores de 220Ω, como na próxima figura.

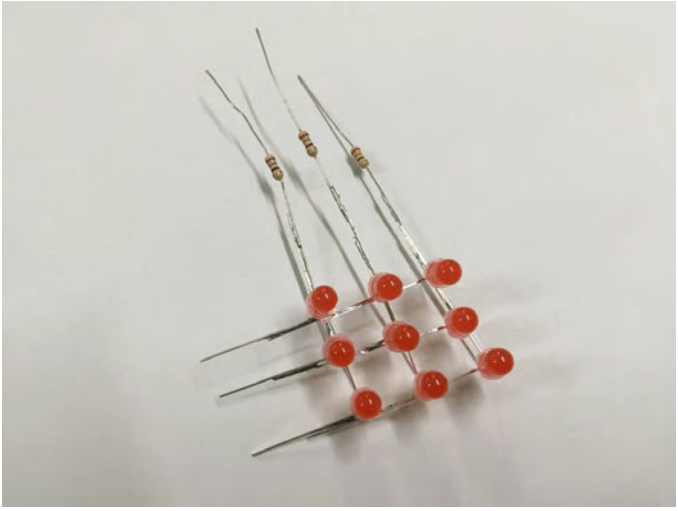


Figura 6.16: Resistores aos terminais positivos da matriz

Agora vamos ao esquema de ligação entre a matriz de LED e o Arduino. Ligue os terminais positivos aos pinos digitais 2, 3 e 4 do Arduino. Use uma sequência, mas considerando que o terminal do meio é sempre o segundo, e que deve estar no pino digital 3.

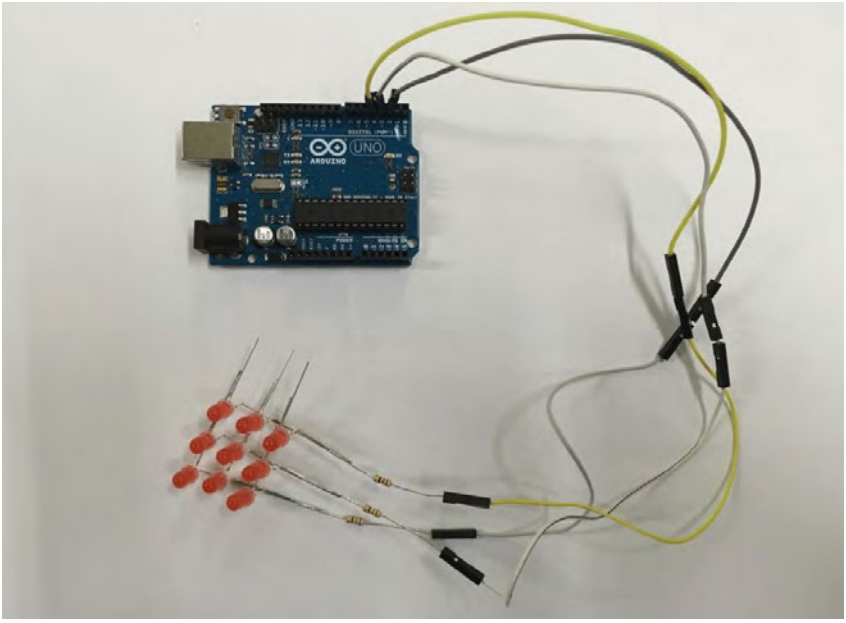


Figura 6.17: Terminais positivos da matriz de LEDs ligados ao Arduino

Continuando, ligue os terminais negativos aos pinos digitais 8, 9 e 10 do Arduino. Use uma sequência, mas considerando que o terminal do meio é sempre o segundo, e que deve estar no pino digital 9.

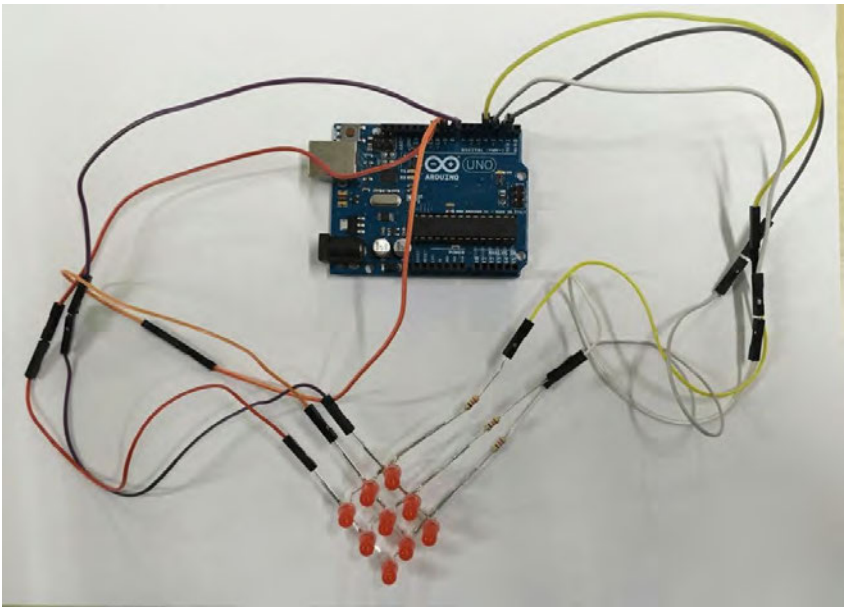


Figura 6.18: Terminais negativos da matriz de LEDs ligados ao Arduino {W=100%}

Para testar se está tudo funcionando, vamos usar o seguinte programa:

```
void setup() {
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
}

void loop() {
  digitalWrite(2,HIGH);
  digitalWrite(3,HIGH);
  digitalWrite(4,HIGH);
  digitalWrite(8,LOW);
  digitalWrite(9,LOW);
  digitalWrite(10,LOW);
}
```

É um programa muitíssimo simples e não demanda qualquer

explicação sobre os comandos usados. O detalhe está em usar os pinos com saída `HIGH` para fornecer tensão aos LEDs, e os pinos com saída `LOW` para funcionarem como terra, fazendo com que a corrente flua e acenda o LED.

Quando temos o pino digital 2 em `HIGH` e o pino digital 8 em `LOW`, temos a conexão positivo-negativo para acender todos os LEDs da primeira linha da matriz de LED. Caso coloquemos o pino digital 2 em `LOW` e/ou o pino digital 8 em `HIGH`, a corrente fluirá ao contrário no LED, que ficará apagado.

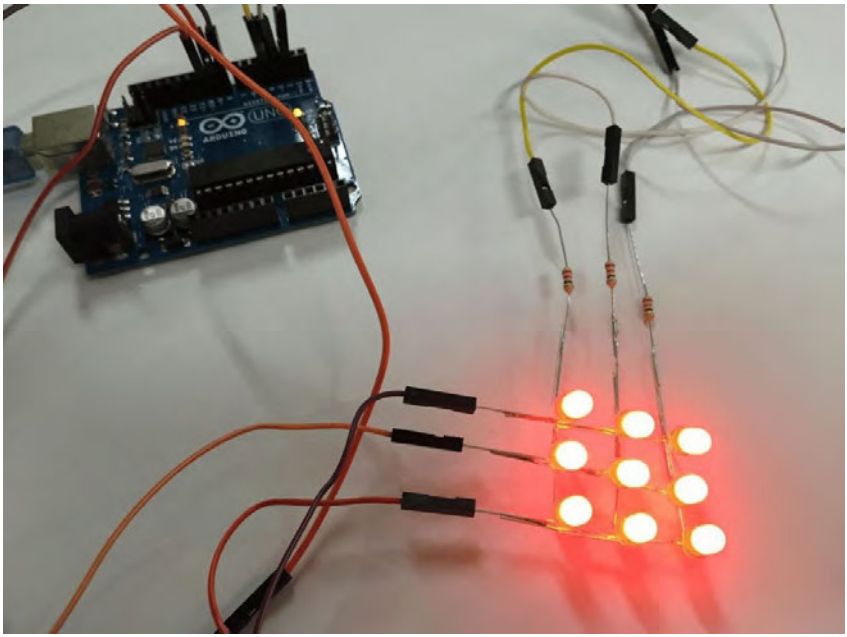


Figura 6.19: Todos os LEDs acesos

Agora vamos alterar esse programa para mostrar exatamente esse funcionamento entre `HIGH` e `LOW` nos pinos digitais, para acender e apagar os LEDs:

```
void setup() {
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
```

```
  pinMode(4,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
}

void loop() {
  for(int i=1;i<=3;i++) {
    for(int j=1;j<=3;j++) {
      acende(i,j);
      delay(500);
    }
  }
}

void acende(int linha,int coluna) {
  if(linha == 1)  digitalWrite(2,HIGH); else digitalWrite(2,LOW);
  if(linha == 2)  digitalWrite(3,HIGH); else digitalWrite(3,LOW);
  if(linha == 3)  digitalWrite(4,HIGH); else digitalWrite(4,LOW);
  if(coluna == 1) digitalWrite(8,LOW);  else digitalWrite(8,HIGH);
  if(coluna == 2) digitalWrite(9,LOW);  else digitalWrite(9,HIGH);
  if(coluna == 3) digitalWrite(10,LOW); else digitalWrite(10,HIGH)
;
}
```

A função `setup()` não mudou, apenas ajusta os pinos digitais como saída, ou seja, `OUTPUT`. No final do programa, criei uma função `void acende(int linha,int coluna)`, que é `void`, então não retornará qualquer valor e receberá dois parâmetros inteiros chamados `linha` e `coluna`.

Essa função acende um LED específico, na linha e coluna desejada, verificando qual é a linha informada, e coloca o valor `HIGH` no respectivo pino digital, para acender o LED. Porém, se a linha não for a informada pelo parâmetro `linha`, o valor no pino digital respectivo será `LOW`, apagando o LED. A coluna funciona da mesma maneira.

Na função `loop()`, criei duas estrutura de repetição `for` para contar as três linhas e as três colunas, e passar exatamente o LED que desejo acender com um intervalo de 500 milissegundos entre acender e apagar cada LED.

Veja um primeiro vídeo teste da matriz de LED, disponível em https://youtu.be/4KByD_i8E1A.

Já que temos como acender um LED específico, vamos criar os números do dado com o programa a seguir:

```
void setup() {
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
}

void loop() {
  um(1);
  dois(1);
  tres(1);
  quatro(1);
  cinco(1);
  seis(1);
}

void acende(int linha,int coluna) {
  if(linha == 1)  digitalWrite(2,HIGH); else digitalWrite(2,LOW);
  if(linha == 2)  digitalWrite(3,HIGH); else digitalWrite(3,LOW);
  if(linha == 3)  digitalWrite(4,HIGH); else digitalWrite(4,LOW);
  if(coluna == 1) digitalWrite(8,LOW);  else digitalWrite(8,HIGH);
  if(coluna == 2) digitalWrite(9,LOW);  else digitalWrite(9,HIGH);
  if(coluna == 3) digitalWrite(10,LOW); else digitalWrite(10,HIGH)
;
}

void um(int t) {
  unsigned long tempo;
  unsigned long tempo_anterior = tempo = millis();
  while(tempo - tempo_anterior < (t * 1000)) {
    tempo = millis();
    acende(2,2);
  }
}
```

```
void dois(int t) {
  unsigned long tempo;
  unsigned long tempo_anterior = tempo = millis();
  while(tempo - tempo_anterior < (t * 998)) {
    tempo = millis();
    acende(1,1);
    delay(1);
    acende(3,3);
    delay(1);
  }
}

void tres(int t) {
  unsigned long tempo;
  unsigned long tempo_anterior = tempo = millis();
  while(tempo - tempo_anterior < (t * 997)) {
    tempo = millis();
    acende(1,1);
    delay(1);
    acende(2,2);
    delay(1);
    acende(3,3);
    delay(1);
  }
}

void quatro(int t) {
  unsigned long tempo;
  unsigned long tempo_anterior = tempo = millis();
  while(tempo - tempo_anterior < (t * 996)) {
    tempo = millis();
    acende(1,1);
    delay(1);
    acende(1,3);
    delay(1);
    acende(3,1);
    delay(1);
    acende(3,3);
    delay(1);
  }
}

void cinco(int t) {
  unsigned long tempo;
  unsigned long tempo_anterior = tempo = millis();
  while(tempo - tempo_anterior < (t * 995)) {
    tempo = millis();
    acende(1,1);
```

```
    delay(1);
    acende(1,3);
    delay(1);
    acende(2,2);
    delay(1);
    acende(3,1);
    delay(1);
    acende(3,3);
    delay(1);
  }
}

void seis(int t) {
  unsigned long tempo;
  unsigned long tempo_anterior = tempo = millis();
  while(tempo - tempo_anterior < (t * 994)) {
    tempo = millis();
    acende(1,1);
    delay(1);
    acende(1,2);
    delay(1);
    acende(1,3);
    delay(1);
    acende(3,1);
    delay(1);
    acende(3,2);
    delay(1);
    acende(3,3);
    delay(1);
  }
}
```

A função `setup()` continua a mesma de antes. A função `loop()` chama as funções que acendem os LEDs específicos para cada número do dado, passando como parâmetro o tempo, em segundos, pelo qual desejamos que esse número seja mantido aceso. E a função `acende` continua a mesma de antes.

Na sequência, temos seis funções do tipo `void`, que não retornam valor e recebem um parâmetro do tipo inteiro, chamado de `t`. Esse parâmetro é a quantidade em segundos que os LEDs devem permanecer acesos.

Não poderemos usar a função `delay` para pausar o programa

com os LEDs acesos, já que precisaremos multiplexá-los. Note que não é possível acender mais de um LED por vez, então no caso de ser necessário mais de um LED para o número, as funções acendem o primeiro, mantêm-no aceso por um milissegundo, apagam-no e acendem o segundo, também mantendo-o aceso por um milissegundo — e assim sucessivamente, pela quantidade de LEDs que forem necessários para formar o número no dado.

Tudo isso é feito usando a função `millis()`, que retorna em milissegundo há quanto tempo o programa está rodando. Para iniciar, declaro duas variáveis com o maior tipo de dado possível: `unsigned long tempo` e `unsigned long tempo_anterior`.

Elas começam com o mesmo valor e, enquanto a diferença entre elas for menor do que o tempo passado pelo parâmetro da função subtraído de um milissegundo a cada LED, elas mantêm a multiplexação entre os LEDs. Ou seja, acende e apaga tão rápido que o olho humano perceberá todos como acesos.

> Você pode ver o resultado desse programa em funcionamento no vídeo disponível em https://youtu.be/i0mJV4R2tfw.

Para terminar o projeto, só nos falta fazer com que o dado sorteie um valor aleatório e mostre o número na matriz de LEDs. Vamos ao último programa:

```
void setup() {
  pinMode(2,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(4,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
  pinMode(10,OUTPUT);
  randomSeed(analogRead(A0));
  int numero = random(1,6);
```

```
  if(numero == 1) um(5);
  if(numero == 2) dois(5);
  if(numero == 3) tres(5);
  if(numero == 4) quatro(5);
  if(numero == 5) cinco(5);
  if(numero == 6) seis(5);
  apaga();
  while(1);
}

void loop() {
}

void apaga() {
  digitalWrite(2,LOW);
  digitalWrite(3,LOW);
  digitalWrite(4,LOW);
  digitalWrite(8,HIGH);
  digitalWrite(9,HIGH);
  digitalWrite(10,HIGH);
}

void acende(int linha,int coluna) {
  if(linha == 1)  digitalWrite(2,HIGH); else digitalWrite(2,LOW);
  if(linha == 2)  digitalWrite(3,HIGH); else digitalWrite(3,LOW);
  if(linha == 3)  digitalWrite(4,HIGH); else digitalWrite(4,LOW);
  if(coluna == 1) digitalWrite(8,LOW);  else digitalWrite(8,HIGH);
  if(coluna == 2) digitalWrite(9,LOW);  else digitalWrite(9,HIGH);
  if(coluna == 3) digitalWrite(10,LOW); else digitalWrite(10,HIGH)
;
}

void um(int t) {
  unsigned long tempo;
  unsigned long tempo_anterior = tempo = millis();
  while(tempo - tempo_anterior < (t * 1000)) {
    tempo = millis();
    acende(2,2);
  }
}

void dois(int t) {
  unsigned long tempo;
  unsigned long tempo_anterior = tempo = millis();
  while(tempo - tempo_anterior < (t * 998)) {
    tempo = millis();
    acende(1,1);
    delay(1);
```

```
    acende(3,3);
    delay(1);
  }
}

void tres(int t) {
  unsigned long tempo;
  unsigned long tempo_anterior = tempo = millis();
  while(tempo - tempo_anterior < (t * 997)) {
    tempo = millis();
    acende(1,1);
    delay(1);
    acende(2,2);
    delay(1);
    acende(3,3);
    delay(1);
  }
}

void quatro(int t) {
  unsigned long tempo;
  unsigned long tempo_anterior = tempo = millis();
  while(tempo - tempo_anterior < (t * 996)) {
    tempo = millis();
    acende(1,1);
    delay(1);
    acende(1,3);
    delay(1);
    acende(3,1);
    delay(1);
    acende(3,3);
    delay(1);
  }
}

void cinco(int t) {
  unsigned long tempo;
  unsigned long tempo_anterior = tempo = millis();
  while(tempo - tempo_anterior < (t * 995)) {
    tempo = millis();
    acende(1,1);
    delay(1);
    acende(1,3);
    delay(1);
    acende(2,2);
    delay(1);
    acende(3,1);
    delay(1);
```

```
    acende(3,3);
    delay(1);
  }
}

void seis(int t) {
  unsigned long tempo;
  unsigned long tempo_anterior = tempo = millis();
  while(tempo - tempo_anterior < (t * 994)) {
    tempo = millis();
    acende(1,1);
    delay(1);
    acende(1,2);
    delay(1);
    acende(1,3);
    delay(1);
    acende(3,1);
    delay(1);
    acende(3,2);
    delay(1);
    acende(3,3);
    delay(1);
  }
}
```

A única coisa que muda, e pouco, é a função `setup()`. Nela usamos o comando `randomSeed(analogRead(A0))` para inicializar a geração de números pseudoaleatórios do Arduino. Para a função `randomSeed`, devemos passar um número que será usado para gerar essa semente de números aleatórios. Como o pino analógico A0 está desconectado de qualquer componente, ele vai retornar um ruído, fazendo com que a semente seja diferente a cada execução do programa.

Na sequência, usamos `int numero = random(1,6)` para armazenar na variável `numero` um valor aleatório entre 1 e 6, gerado pela função `random`. Depois é só usar uma estrutura de seleção `if` para verificar qual é o número e chamar a função do número correspondente.

Cada número será mantido aceso por 5 segundos e, depois, a matriz apagará, indicando o final da execução, já que a função

`loop()` está vazia e não tem qualquer comando para executar. Para realizar um novo sorteio do dado, basta resetar o Arduino.

> Você poderá ver esse projeto finalizado em funcionamento em https://youtu.be/TzaHFa9i9qQ.

Boas jogadas!

6.3 DESAFIO

Acho que o principal desafio deste projeto é colocá-lo em uma caixa que acomode os LEDs à mostra, de forma que possa ser usado efetivamente como um dado.

Isso talvez requeira que você retorne ao projeto nº 01 (*capítulo 2*) e crie seu próprio Arduino, para que tudo fique pequeno o suficiente para que o dado tenha, além dos LEDs, o Arduino e uma bateria de 9V. É interessante também fazer os LEDs piscarem em alguma sequência quando nenhum número esteja sendo sorteado.

Bom divertimento com seus jogos de tabuleiro! E, por falar em jogos, que tal criarmos nosso próprio videogame? Bem simples, claro, mas suficientemente jogável em uma TV com uma entrada analógica do tipo RCA. Vamos nessa?

CAPÍTULO 7

PROJETO Nº 06 — CRIANDO UM VIDEOGAME PARA VOCÊ

Quando falo em criar um videogame usando Arduino, logo todo mundo vai pensando em jogos dos consoles atuais. Calma lá! Com Arduino, por enquanto, não é possível ir tão longe. Mas dá para reproduzir alguns jogos clássicos da Atari, por exemplo, como o Pong ou Space Invaders.

Nesse projeto, a ideia é criar um jogo que não seja exatamente uma cópia de qualquer jogo existente, mas testar as capacidades do Arduino de gerar imagens na televisão e na criatividade de criarmos um enredo e algum movimento.

7.1 MATERIAIS UTILIZADOS NESSE PROJETO

- 1 x Arduino UNO
- 1 x Prot-o-board
- 1 x Resistor de 470Ω
- 1 x Resistor de 1KΩ
- 1 x Potenciômetro de 10KΩ

- 1 x Conector RCA macho
- 1 x Ferro de solda
- Estanho para solda
- Cabo com pelo menos duas vias (fios)
- Fios diversos

7.2 DESENVOLVENDO O PROJETO

Pensei no seguinte jogo: uma nave espacial, viajando pelo espaço profundo em busca de vida em outros planetas, enfrenta uma área de asteroides que vêm em sua direção. Você, o superpiloto dessa nave, tem a missão de salvar toda a tripulação demonstrando o quão bom é em desviar desses asteroides!

E como construir tudo isso? **Fácil!** Eis do que você vai precisar: um resistor de 1KΩ, um resistor de 470Ω, um conector RCA macho, um potenciômetro de 10KΩ, um cabo com 1,5 metros de pelo menos duas vias, alguns fiozinhos para ligar tudo, uma prot-o-board e, claro, um Arduino.

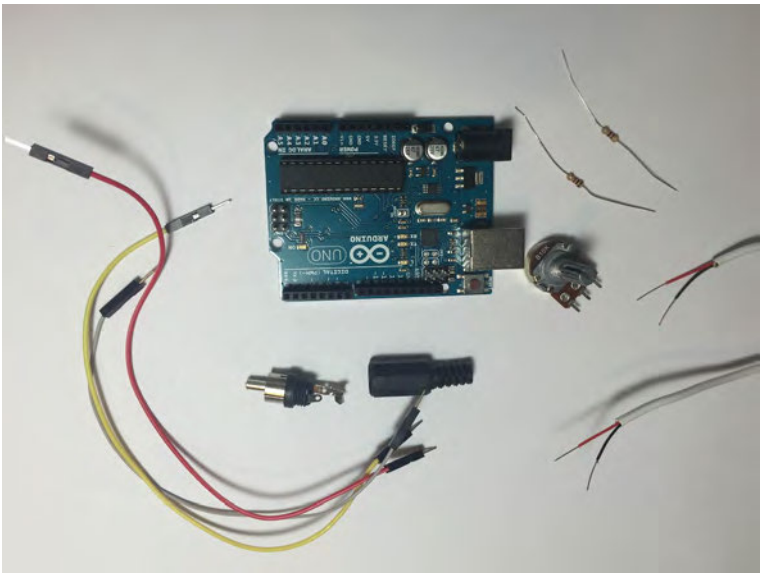


Figura 7.1: Componentes necessários para criar nosso videogame

Pegue os dois resistores e solde um terminal de cada um a uma das vias do cabo. Você verá na figura seguinte que usei o fio vermelho para isso.

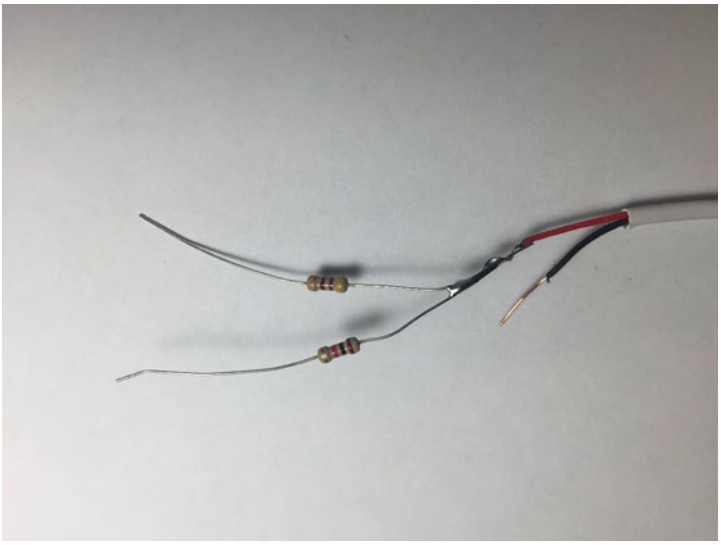


Figura 7.2: Resistores em um dos fios do cabo

Do outro lado do cabo, solde o mesmo fio que recebeu os resistores ao centro do conector RCA. No meu caso, usei o fio vermelho para os resistores, portanto, do outro lado do cabo, o fio vermelho será soldado ao centro do conector RCA. O outro fio deve ser soldado ao outro terminal (externo) do conector RCA.

Figura 7.3: Fios soldados ao conector RCA

Agora ligue o outro terminal do resistor de 470Ω ao pino digital 7, e o outro terminal do resistor de 1KΩ ao pino digital 9 do Arduino. O outro fio, desse mesmo lado dos resistores, vai ao pino GND do Arduino.

Figura 7.4: Conexão com o Arduino

O conector RCA vai na entrada de vídeo da sua televisão. Pronto! Falei que seria fácil, todas as conexões com o Arduino e a TV estão feitas! Vamos à programação.

Primeiro você terá de obter a biblioteca TVout, que está disponível em https://github.com/fbryan/tvout. Para instalá-la, siga os procedimentos do projeto nº 02 (*capítulo 3*).

Agora, vamos ao programa:

```
#include <TVout.h>
#include <video_gen.h>

TVout tv;
int led = 13;

void setup() {
 pinMode(led,OUTPUT);
 if(tv.begin(NTSC,128,96) == 0) {
  digitalWrite(led,HIGH);
```

```
 } else {
  digitalWrite(led,LOW);
 }
 delay(1000);
 tv.draw_rect(0,0,71,55,WHITE,BLACK);
 tv.draw_line(0,0,10,10,WHITE);
 tv.draw_circle(36,28,10,WHITE);
 tv.set_pixel(36,28,WHITE);
}

void loop() {
}
```

Já no começo temos a importação das duas bibliotecas necessárias para gerar imagens na TV. Fazemos isso com `#include <TVout.h>` e `#include <video_gen.h>`.

O próximo passo é criar um objeto do tipo `TVout` chamado `tv`, que será usado para executarmos os métodos disponíveis para criar imagens na sua TV. Também criei uma variável chamada `led`, do tipo inteiro, que recebe o valor `13`, para indicar o pino digital em que estará um LED usado, apenas para indicar o bom funcionamento de tudo.

Na função `setup()`, ajustamos o pino digital do LED como saída usando `pinMode(led,OUTPUT)`. Em seguida, usamos `tv.begin(NTSC,128,96)` para inicializar a geração de imagens com o padrão NTSC e resolução de 128x96 pixels.

Os padrões suportados são NTSC ou PAL e a melhor resolução na prática precisa ser descoberta empiricamente. Essa função retorna `0` caso a inicialização seja normal; `1` caso a resolução horizontal não seja múltipla de 8; `2` caso a resolução vertical não seja múltipla de 8; e `3` caso não haja memória suficiente no Arduino. Por isso, nós a colocamos dentro de uma estrutura de seleção `if`. Caso o retorno seja de inicialização normal, acendemos o LED do pino digital 13 e seguimos em frente. Caso seja retornado qualquer código de erro, o LED ficará apagado, indicando algo

errado.

Com `delay(1000)`, damos um segundo para que a TV sincronize com o sinal. Para exemplificar e testar a geração de imagem, usamos os quatro métodos mais comuns:

- `draw_rect` para desenhar retângulos;
- `draw_line` para desenhar linhas;
- `draw_circle` para desenhar círculos;
- `set_pixel` para desenhar apenas um pixel.

O método `draw_rect` recebe como parâmetros a posição do canto superior esquerdo, a posição do canto inferior direito, a cor da linha e a cor do preenchimento. Então, com `tv.draw_rect(0,0,71,55,WHITE,BLACK)`, desenhamos um retângulo iniciado na coluna e linha 0 até a coluna 71 e linha 55. A linha será branca e o preenchimento preto.

Com `tv.draw_line(0,0,10,10,WHITE)`, será desenhada uma linha entre os pixels nas posições `0,0` e `10,10` na cor branca. Já `tv.draw_circle(36,28,10,WHITE)` desenhará um círculo nas posições `36,28` com raio de 10 pixels e linha na cor branca. Com `tv.set_pixel(36,28,WHITE)`, desenhará o pixel na posição `36,28` com a cor branca.

As cores possíveis são `WHITE` para branco, `BLACK` para preto e `INVERT` para inverter a cor em relação ao fundo. Faça upload do código para seu Arduino e, com ele ligado à TV, você deverá ver uma figura como a que segue.

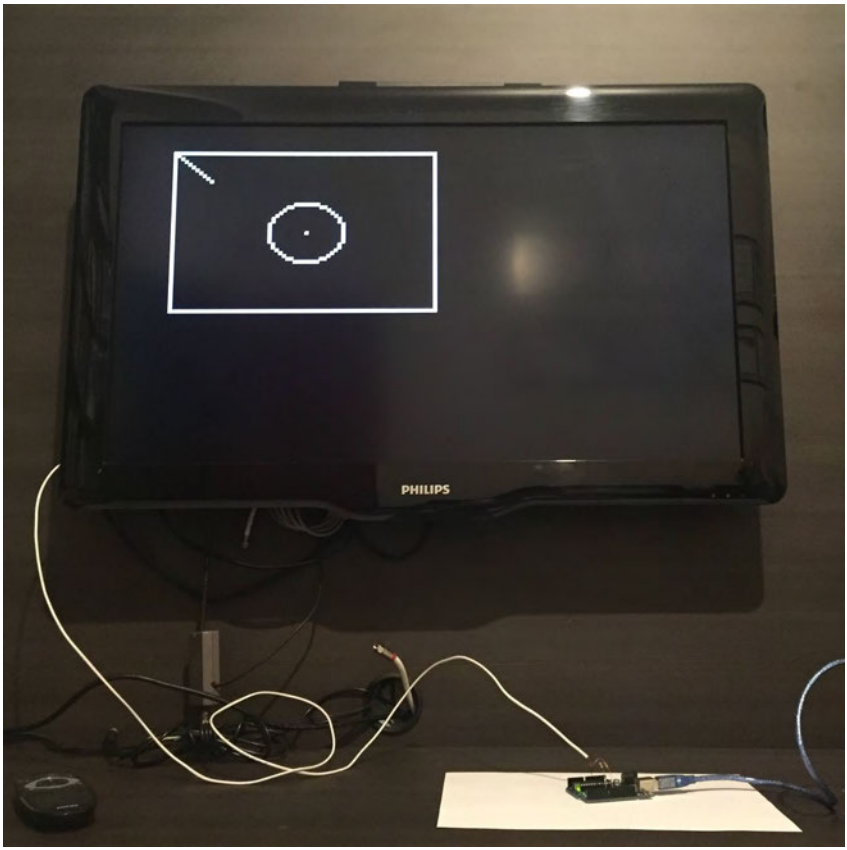


Figura 7.5: Resultado do programa na TV

Ok, imagem na TV! Hora de começarmos a criar nosso jogo. O primeiro passo será fazer os asteroides saírem da esquerda da tela e irem para a direita. O programa é o que segue:

```
#include <TVout.h>
#include <video_gen.h>

TVout tv;
int led = 13;
int x = 0;
int y[5];

void setup() {
 pinMode(led,OUTPUT);
```

```
 if(tv.begin(NTSC,128,96) == 0)
  digitalWrite(led,HIGH);
 else
  digitalWrite(led,LOW);
 delay(1000);
 randomSeed(analogRead(A5));
 for(int i=0;i<5;i++) y[i] = random(0,tv.hres());
}

void loop() {
  for(int i=0;i<5;i++) tv.set_pixel(x,y[i],WHITE);
  delay(10);
  for(int i=0;i<5;i++) tv.set_pixel(x,y[i],BLACK);
  delay(10);
  if(x > tv.hres()) {
    for(int i=0;i<5;i++) y[i] = random(0,tv.hres());
    x = 0;
  } else {
    x++;
  }
}
```

Como você pode notar, a importação e a inicialização da geração de imagens são as mesmas. Portanto, não vou me ocupar com elas novamente.

A variável inteira `x` será a coluna na qual os asteroides deverão estar, e o vetor `y` armazenará a linha de cada asteroide. Note o tamanho `5` do vetor, ou seja, teremos apenas 5 asteroides por vez.

Na função `setup()`, usamos `randomSeed(analogRead(A5))` para inicializar a semente de números aleatórios. Passamos como parâmetro a leitura do pino analógico A5, que está desconectado e gerará um ruído aleatório, fazendo com que a semente seja diferente a cada inicialização do programa.

Com `for(int i=0;i<5;i++) y[i] = random(0,tv.hres())`, sorteamos uma linha para cada asteroide e a guardamos no vetor `y`. Dessa forma, manteremos a trajetória do asteroide o tempo todo pela mesma linha, mas variando apenas as colunas.

Na função `loop()`, usamos uma estrutura de repetição `for(int i=0;i<5;i++) tv.set_pixel(x,y[i],WHITE)` para acender os pixels que serão nossos asteroides. Em seguida, damos 10 milissegundos para que fiquem na tela. Na sequência, fazemos a mesma coisa, mas para apagá-los, dessa forma teremos a sensação de movimento.

Veja trecho a seguir:

```
if(x > tv.hres()) {
  for(int i=0;i<5;i++) y[i] = random(0,tv.hres());
  x = 0;
} else {
  x++;
}
```

Nele, verificamos se a coluna (variável `x`) onde está o asteroide é maior do que a resolução horizontal da tela. Caso seja, é preciso sortear uma nova posição para as linhas dos asteroides e voltar a variável `x` para 0. Ou seja, os asteroides voltarão à esquerda da tela e partirão para cima da nave novamente em novas posições de ataque.

Caso a coluna não seja maior que a resolução horizontal da tela, acrescenta-se um à variável `x` para que o asteroide avance para à direita.

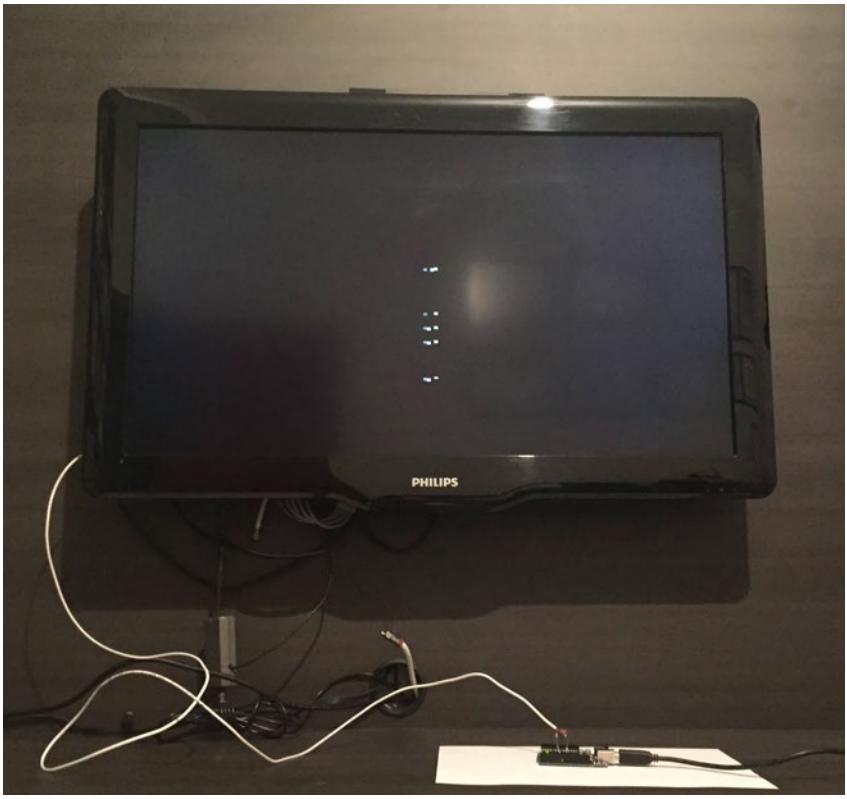


Figura 7.6: Asteroides na tela

Você poderá ver um vídeo desse programa funcionando em https://youtu.be/0cNT9CwF3YY.

Agora falta fazermos o controle e a nave!

Para o controle, conecte o potenciômetro ao prot-o-board e um dos terminais laterais dele ao pino 5V, o central ao pino analógico A0 e o terminal do potenciômetro da outra lateral ao pino GND do Arduino.

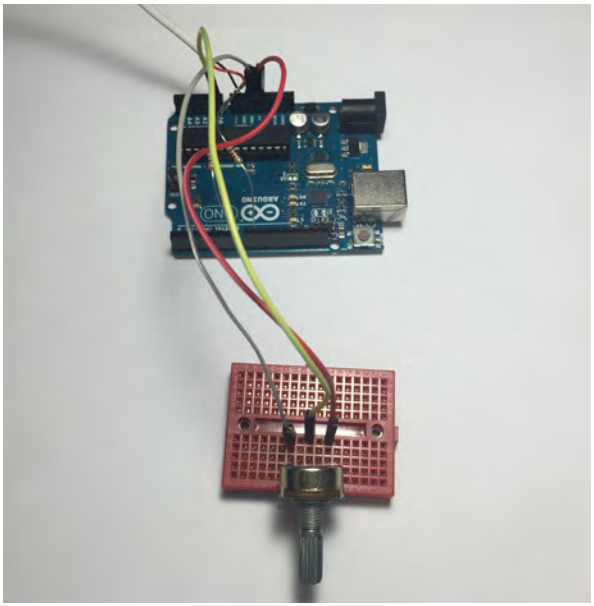


Figura 7.7: Potenciômetro ligado ao Arduino

E para terminar, o programa final do jogo:

```
#include <TVout.h>
#include <video_gen.h>

TVout tv;
int led = 13;
int x = 0;
int y[5];
int x_nave;
int y_nave;

void setup() {
 pinMode(led,OUTPUT);
 if(tv.begin(NTSC,128,96) == 0)
  digitalWrite(led,HIGH);
 else
  digitalWrite(led,LOW);
 delay(1000);
 randomSeed(analogRead(A5));
 for(int i=0;i<5;i++) y[i] = random(0,tv.vres());
 x_nave = tv.hres()-1;
 y_nave = tv.vres()/2;
}
```

```
void loop() {
  int ctrl = map(analogRead(A0),0,1024,0,tv.vres());
  tv.draw_line(x_nave,ctrl-2,x_nave,ctrl+2,WHITE);
  delay(10);
  tv.draw_line(x_nave,ctrl-2,x_nave,ctrl+2,BLACK);
  for(int i=0;i<5;i++) tv.set_pixel(x,y[i],WHITE);
  delay(10);
  for(int i=0;i<5;i++) tv.set_pixel(x,y[i],BLACK);
  if(x > tv.hres()) {
    for(int i=0;i<5;i++) y[i] = random(0,tv.hres());
    x = 0;
  } else {
    x++;
  }
  if(x >= x_nave) {
    for(int i=0;i<5;i++) {
      if((y[i] >= ctrl-2) && (y[i] <= ctrl+2)) {
        tv.clear_screen();
        for(int d=0;d<tv.hres();d++) {
          tv.draw_circle(tv.hres()/2,tv.vres()/2,d,WHITE);
          delay(10);
        }
        delay(15000);
        for(int i=0;i<5;i++) y[i] = random(0,tv.hres());
        tv.clear_screen();
      }
    }
  }
}
```

Como de costume, esse programa é uma alteração do anterior, então vamos nos conter apenas ao que mudou. As variáveis `x_nave` e `y_nave` servirão para armazenar as posições `x` e `y` da nave.

Na função `setup()`, a posição da nave é um pixel antes do valor horizontal máximo definido por `x_nave = tv.hres()-1`, e no meio da tela verticalmente, definido por `y_nave = tv.vres()/2`.

Na função `loop()`, a mágica toda acontece! O movimento dos asteroides será o mesmo, mas o movimento da nave será definido pela leitura do valor do potenciômetro feita com `int ctrl =`

`map(analogRead(A0),0,1024,0,tv.vres())` . A variável inteira `ctrl` (controle) armazenará a leitura do pino analógico A0, que retorna valores entre 0 e 1024, mas que serão convertidos proporcionalmente para entre 0 e a resolução total da tela — conversão esta feita pela função `map` .

A nave será uma linha desenhada com `tv.draw_line(x_nave,ctrl-2,x_nave,ctrl+2,WHITE)` . Note que o seu tamanho é definido pela subtração de 2 no limite superior, e soma de 2 no limite inferior da linha. Quando a leitura do potenciômetro variar, a posição da nave também variará.

A verificação da colisão dos asteroides com a nave está no seguinte bloco:

```
if(x >= x_nave) {
    for(int i=0;i<5;i++) {
      if((y[i] >= ctrl-2) && (y[i] <= ctrl+2)) {
        tv.clear_screen();
        for(int d=0;d<tv.hres();d++) {
          tv.draw_circle(tv.hres()/2,tv.vres()/2,d,WHITE);
          delay(10);
        }
        delay(15000);
        for(int i=0;i<5;i++) y[i] = random(0,tv.hres());
        tv.clear_screen();
      }
    }
  }
```

A variável `x` é a posição do asteroide, e `x_nave` é a posição da nave. Se o asteroide tiver uma posição `x` maior ou igual a da nave, é que eles estão na mesma coluna, então pode estar colidindo. Basta verificar se estão na mesma linha.

Caso a linha do asteroide estiver dentro do tamanho da nave, verificado com `if((y[i] >= ctrl-2) && (y[i] <= ctrl+2))` , e a colisão seja detectada, a sequência será: limpar a tela com `tv.clear_screen()` e fazer com que um círculo cresça a partir do

centro da tela, desenhado com `tv.draw_circle(tv.hres()/2,tv.vres()/2,d,WHITE)`, representando uma explosão catastrófica.

Quinze segundos se passarão graças ao `delay(15000)`, novas posições para os asteroides serão sorteadas, a tela será limpa novamente e o jogo recomeçará.

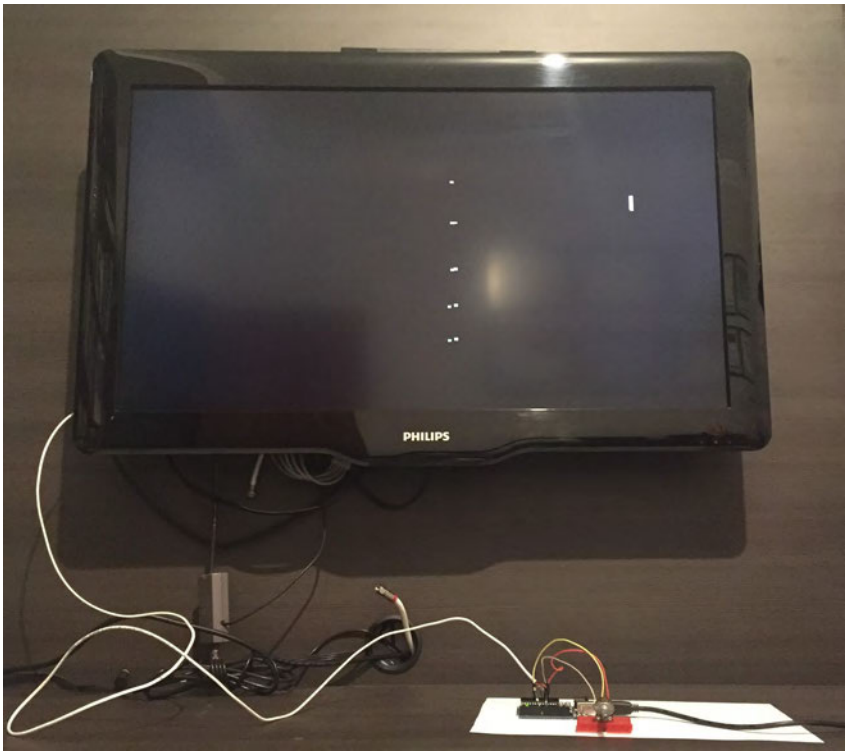


Figura 7.8: Asteroides em direção à nave

Agora é só se divertir!

Um vídeo do jogo em funcionamento poderá ser visto em https://youtu.be/UDYPNbFKZQ0.

7.3 DESAFIO

O limite é o céu! Aliás, simplesmente não há limites! Você pode colocar tudo dentro de uma caixa e criar seu próprio console. O microcontrolador pode estar dentro de um cartucho, que é conectado ao Arduino como se fosse o console. Dessa forma, você pode gravar vários programas diferentes em vários microcontroladores diferentes, e ter jogos em formato de cartucho.

Para a criação de jogos, há uma infinidade de possibilidades com enredos, movimentos, desenhos, tudo diferente do que qualquer um jamais pensou, ou até mesmo reproduzindo jogos antigos conhecidos e amados. Contar pontos também é sempre uma boa ideia! Um placar cria a possibilidade de registrar recordes, e desafiar seus familiares e amigos a serem mais ágeis ou mais rápidos.

Mas nem tudo é diversão, é preciso monitorar e controlar algumas coisas para evitar gastos desnecessários, como por exemplo, eletricidade. Por isso, o próximo projeto é de um alarme em que a sugestão é colocar na geladeira, de forma a não permitir que esqueçam a sua porta aberta. Vamos lá!

CAPÍTULO 8

PROJETO Nº 07 — ALARME DE GELADEIRA COM MONITORAMENTO DE ABERTURA DA PORTA

Durante a construção dos projetos, sempre há aquela paradinha para alguns petiscos, porque ninguém é de ferro, né?! Nessas idas e vindas, acabei por esquecer a geladeira aberta. Bronca na certa!

O mais interessante é que a porta ficou levemente entreaberta, mal dando para perceber que tinha ficado assim. E como minha geladeira não tem alarme embutido para esses casos, resolvi criar um para evitar que isso volte a acontecer.

8.1 MATERIAIS UTILIZADOS NESSE PROJETO

- 1 x Arduino UNO
- 1 x Prot-o-board
- 1 x Sensor magnético para alarmes (Reed-switch)
- 1 x Resistor 1KΩ 1/4W
- 1 x Buzzer
- 1 x Fonte para telefone celular 5V 1A USB

- 1 x Cabo USB
- Fios diversos

8.2 DESENVOLVENDO O PROJETO

Entre as várias possibilidades que examinei, está um sensor de luminosidade que, caso a porta fique aberta e a luz da geladeira acesa, toque um alarme. Mas logo abandonei essa ideia porque, no meu caso, a porta ficou aberta, mas a luz estava apagada, pois não estava aberta o suficiente para isso. Também pensei que a luz da geladeira, apesar de que hoje em dia isso é bem difícil pelo uso de LED, poderia queimar ou por qualquer outro problema não acender mais.

Outra ideia que tive foi acessar o sensor da porta da geladeira, que acende a luz quando a porta abre. Mas também abandonei pelos motivos já apresentados e porque abrir a geladeira, além de ser difícil, poderia fazer com que uma possível garantia fosse perdida.

O melhor é algo não invasivo, então escolhi usar um sensor magnético, daqueles usados em sistemas de alarmes para detectar abertura de portas e janelas. Na verdade, aquele sensor é um conjunto chave de Reed (*Reed-switch*) e imã.

Bem resumidamente, um Reed-switch é composto por uma ampola de vidro com duas lâminas metálicas que, sob a ação magnética de um imã, por exemplo, sofrem atração e entram em contato entre elas, ligando o circuito. Quando o imã é afastado, elas voltam às suas posições originais, desligando o circuito.

Um esquema básico de uma chave Reed é a figura seguinte:

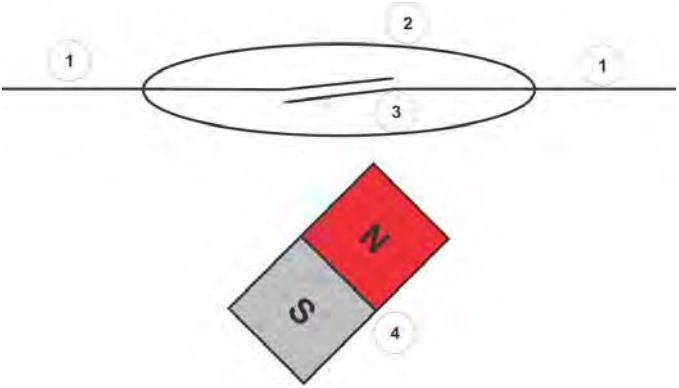


Figura 8.1: Esquema interno de uma chave de Reed

No número 1, temos os terminais; o número 2 é a ampola de vidro; no número 3 temos os contatos, separados por uma pequena distância; o número 4 representa um imã que, quando aproximado da chave, fará com que os contatos (3) toquem entre si, ligando o circuito, e quando o imã for afastado, os contatos voltarão à posição original, desligando o circuito.

Como já mencionei, dentro do sensor magnético de alarme, o que temos é uma chave Reed (Reed-switch). Então os fios que saem do sensor são os contatos da chave.

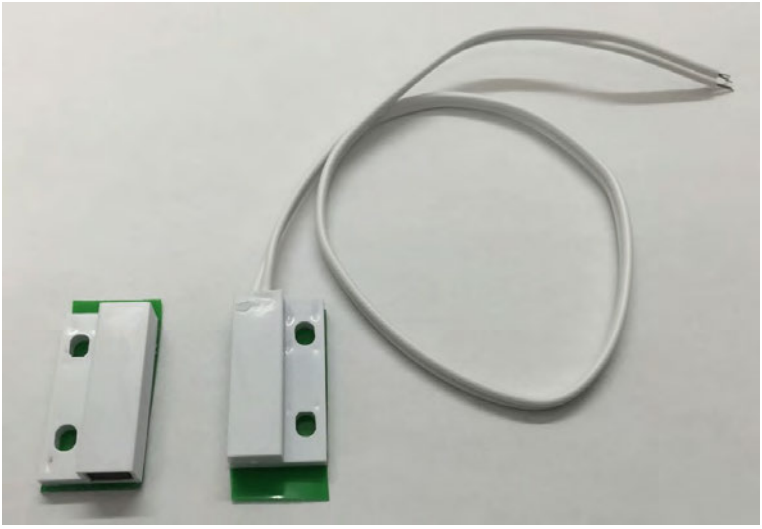


Figura 8.2: Um sensor magnético de alarme — no interior, há uma chave Reed

Tudo isso faz da ligação do sensor magnético ao Arduino ser o mesmo de um botão comum. Ligue um resistor de 1KΩ entre o pino digital 2 e o GND, depois ligue um dos fios do sensor ao pino digital 2 e o outro ao pino 5V. Veja como fica na figura:

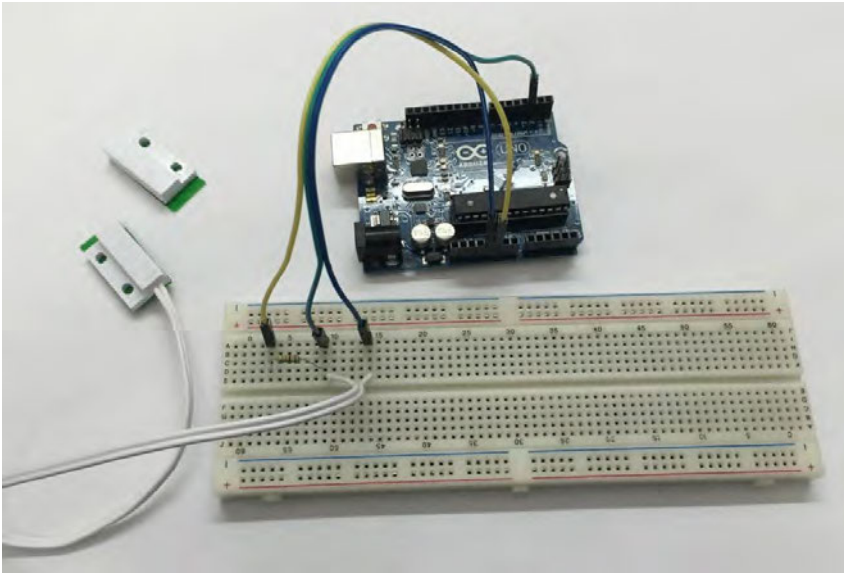


Figura 8.3: A ligação do sensor magnético ao Arduino

O resistor funcionará como um resistor PULL-DOWN externo. Note que, no projeto nº 3 (*capítulo 4*), usamos o resistor PULL-UP interno, mas nesse caso resolvi mostrar como fazer isso sem usá-lo.

O PULL-DOWN garantirá o valor `LOW` no pino digital 2 enquanto o botão não estiver pressionado (aberto). E garantirá o valor `HIGH` quando ele estiver fechado, ou seja, pressionado.

Para testar, vamos criar um código que, quando os contatos da chave Reed estiverem fechados (ou seja, quando o imã está próximo), o LED do pino digital 13 do Arduino ficará aceso. Quando você afastar o imã, os contatos da chave Reed abrirão e o LED apagará. Com isso, poderemos verificar na prática o funcionamento de tudo isso.

```
int led = 13;
int sensor = 2;

void setup() {
  pinMode(led,OUTPUT);
```

```
  pinMode(sensor,INPUT);
}

void loop() {
  if(digitalRead(sensor) == HIGH) {
    digitalWrite(led,HIGH);
  } else {
    digitalWrite(led,LOW);
  }
}
```

A variável inteira `led` indica em qual pino digital está ligado o LED que acionaremos. A variável `sensor` indica em qual pino digital está ligada a nossa chave Reed. Na função `setup()`, apenas ajustamos o pino digital 13 (do LED) como saída, e o pino digital 2 (do sensor) como entrada.

Na função `loop()`, usamos `digitalRead(sensor)` para ler o estado do pino digital do botão, que é o pino 2. Ela vai retornar os valores `LOW` (`0`) caso a chave esteja aberta (desligada), ou `HIGH` (`1`) caso a chave esteja fechada (ligada). Sabendo disso, basta usar uma estrutura de seleção `if(digitalRead(sensor) == HIGH)` para saber se a chave está fechada e, caso esteja, ligar o LED com `digitalWrite(led,HIGH)`; caso contrário, apagamos o LED com `digitalWrite(led,LOW)`.

Depois de compilar e fazer o upload do programa para seu Arduino, você pode testar o funcionamento do programa e do sensor aproximando o imã do sensor. Quando o imã estiver próximo o suficiente, o LED do pino digital 13 do Arduino deverá acender; caso contrário deve permanecer apagado. Note que a distância em que o imã tem influência sobre o sensor pode variar, mas normalmente é de aproximadamente 2 centímetros.

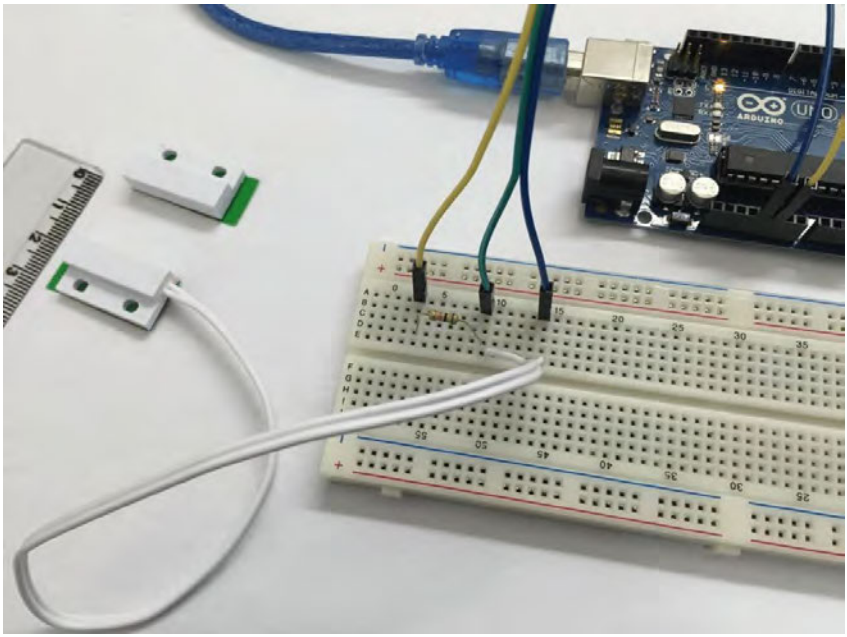


Figura 8.4: Imã influenciando o sensor e acendendo o LED do pino digital 13

Agora vamos mudar nosso código para capturar o tempo em que a porta da geladeira ficou aberta:

```
int led = 13;
int sensor = 2;

int ctrl_abre = 0;
int ctrl_fecha = 0;

unsigned long tempo_abrir = 0;
unsigned long tempo_fechar = 0;

void setup() {
  Serial.begin(9600);
  pinMode(led, OUTPUT);
  pinMode(sensor, INPUT);
}

void loop() {
  if (digitalRead(sensor) == HIGH) {
    digitalWrite(led, HIGH);
    if (ctrl_abre == 0) {
```

```
      tempo_fechar = millis();
      ctrl_abre = 1;
      ctrl_fecha = 0;
      Serial.print("Porta ficou aberta por ");
      Serial.print((tempo_fechar - tempo_abrir) / 1000);
      Serial.println(" segundos");
    }
  } else {
    digitalWrite(led, LOW);
    if (ctrl_fecha == 0) {
      tempo_abrir = millis();
      ctrl_fecha = 1;
      ctrl_abre = 0;
    }
  }
}
```

Declarei duas variáveis do tipo inteiro para controlar se a porta está aberta ou fechada. As variáveis têm nomes óbvios, sendo que a que controla se a porta está aberta é a variável `ctrl_abre`, e a que controla a porta fechada é a `ctrl_fecha`. Se `ctrl_abre` estiver com valor zero, é que a porta está fechada; se estiver com valor 1, indica que a porta está aberta.

Se a variável `ctrl_fecha` estiver com valor zero, a porta está aberta; se estiver com o valor 1, indica que está fechada. Funcionam ao contrário uma da outra.

Também declarei duas variáveis para contar o tempo em que a porta ficou aberta. São variáveis do tipo `unsigned long`. A variável `tempo_abrir` armazenará o tempo no qual a porta foi aberta, e a variável `tempo_fechar` capturará o instante em que a porta fechou.

Na função `setup()`, além do que já vimos anteriormente, iniciei a comunicação serial com o computador a 9600 bauds. Na função `loop()`, lemos o sensor como fizemos no código anterior e controlamos o LED do pino digital 13 da mesma forma. Porém, adicionei uma estrutura de controle para saber quando a porta está aberta e fechada. Para detectar o movimento da porta, usamos as

variáveis `ctrl_abre` e `ctrl_fecha`, como expliquei anteriormente.

O funcionamento ficou assim: quando o programa inicia, presume-se que a porta está fechada. Então, o LED do pino digital 13 acende e a variável `tempo_fechar` recebe o tempo desde o início da execução do programa através da função `millis()`. A variável `ctrl_abre` recebe 1, indicando falso (ou seja, a porta não está aberta), e a `ctrl_fecha` recebe 0, indicando verdadeiro, já que a porta está fechada.

O cálculo de quanto tempo a porta ficou aberta está na linha `Serial.print((tempo_fechar - tempo_abrir) / 1000)`, que faz a subtração do instante em que a porta fechou e o instante em que abriu. Divide-se tudo por 1000, pois a medição é em milissegundos e queremos o resultado em segundos. O texto com o tempo é enviado ao Monitor Serial.

Dê uma atenção à função `millis()`. Ela retorna o tempo em milissegundo em que o programa está em execução. Ela estoura a capacidade da variável mais ou menos a cada 50 dias, voltando para o zero. Então, pelo seu retorno, antes que isso aconteça, podemos contar o tempo sem mesmo usar um relógio interno ou externo.

Agora, quando a porta é aberta, o sensor desligado, o LED do pino digital 13 apaga, e a variável `tempo_abrir` recebe o tempo de execução até esse instante. A variável `ctrl_fecha` recebe 1 e `ctrl_abre` recebe 0, indicando que a porta está aberta.

Para testar, compile e carregue o programa no seu Arduino e abra o Monitor Serial. Afaste o imã do sensor para simular a abertura da porta e, depois de alguns segundos, aproxime-o de volta, simulando o fechamento da porta da geladeira.

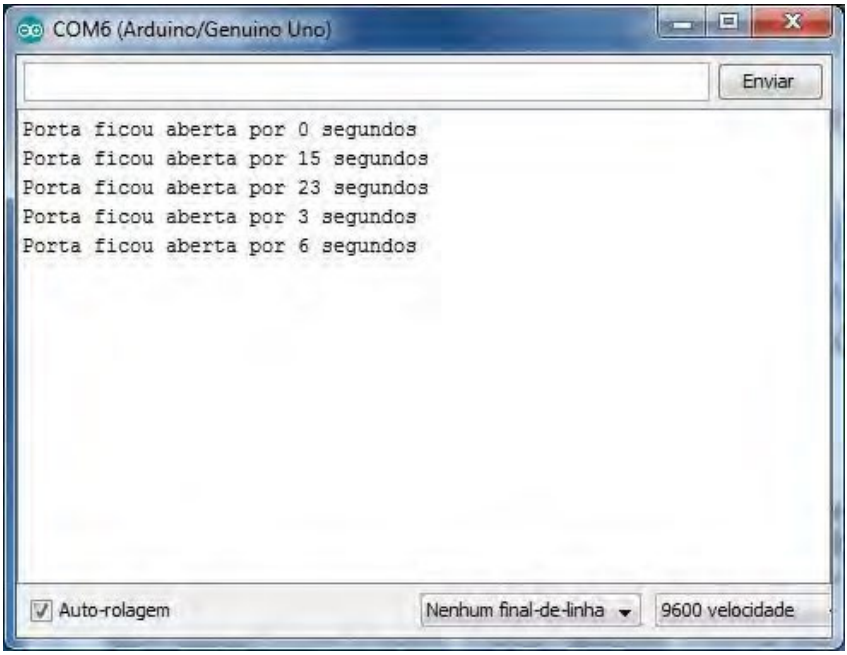


Figura 8.5: Resultado da simulação de abertura e fechamento da porta

Agora faltam apenas três coisas:

1. Incluir um buzzer para que um alarme dispare se a porta ficar aberta por muito tempo.
2. Tornar possível guardar quantas vezes a porta da geladeira foi aberta desde a última inicialização e o tempo de abertura máximo. Claro que também precisaremos de um meio para ler essas informações.
3. Instalar em uma geladeira de verdade!

Vamos pelo buzzer! Para economizar espaço, ligaremos o buzzer diretamente ao Arduino. Poderíamos colocá-lo na prot-o-board junto com todo o resto, mas não vamos deixar que nosso projeto vire um emaranhado de fios.

Ligue o terminal positivo do buzzer ao pino digital 11 e o

terminal negativo ao GND do Arduino.

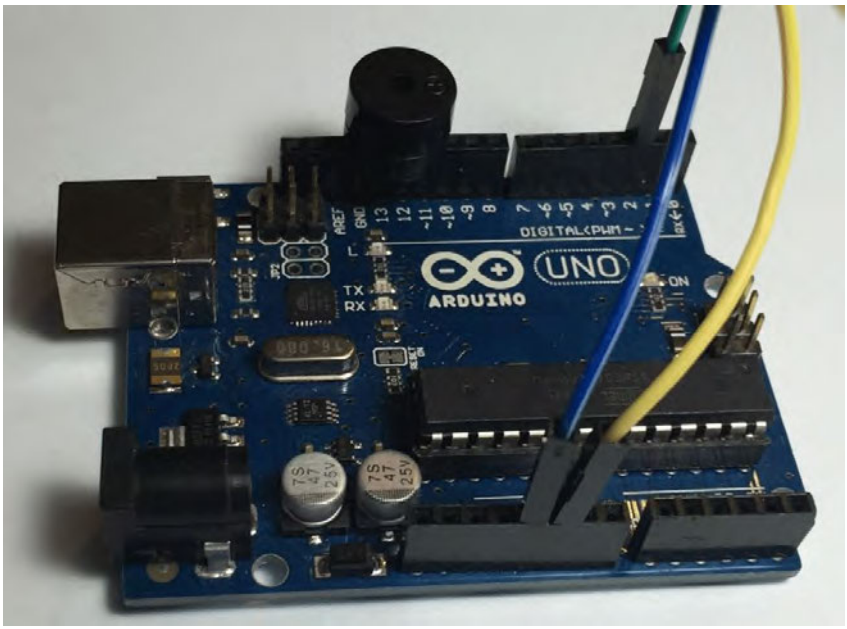


Figura 8.6: Buzzer ligado diretamente ao Arduino

No código, vou prever que o tempo máximo que a porta da geladeira poderá ficar aberta antes de o buzzer disparar será de 30 segundos. Mas como você verá, será fácil ajustar esse parâmetro.

```
int tempo = 0;
int tempo_max = 30;

int led = 13;
int sensor = 2;
int buzzer = 11;

int ctrl_abre = 0;
int ctrl_fecha = 0;

unsigned long tempo_abrir = 0;
unsigned long tempo_fechar = 0;

void setup() {
  Serial.begin(9600);
  pinMode(buzzer,OUTPUT);
```

```
  pinMode(led, OUTPUT);
  pinMode(sensor, INPUT);
}

void loop() {
  if (digitalRead(sensor) == HIGH) {
    digitalWrite(led, HIGH);
    if (ctrl_abre == 0) {
      tempo_fechar = millis();
      ctrl_abre = 1;
      ctrl_fecha = 0;
      Serial.print("Porta ficou aberta por ");
      Serial.print((tempo_fechar - tempo_abrir) / 1000);
      Serial.println(" segundos");
    }
  } else {
    digitalWrite(led, LOW);
    tempo = millis() / 1000;
    if(tempo - (tempo_fechar / 1000) > tempo_max) {
      digitalWrite(buzzer,HIGH);
      delay(150);
      digitalWrite(buzzer,LOW);
      delay(150);
    }
    if (ctrl_fecha == 0) {
      tempo_abrir = millis();
      ctrl_fecha = 1;
      ctrl_abre = 0;
      tempo = 0;
    }
  }
}
```

Focando apenas nas mudanças, repare que declarei duas novas variáveis do tipo inteiro logo no começo do programa. São as variáveis `tempo` , inicializada com zero, que servirá para contagem do tempo até disparar o buzzer; e `tempo_max` , inicializada com 30, que indica o tempo máximo que deve ser contado, em segundos, até disparar o buzzer.

Declarei também uma variável `buzzer` que é inicializada com 11, indicando em qual pino digital do Arduino o buzzer está ligado. Na função `setup()` , usei `pinMode(buzzer,OUTPUT)` para setar o pino digital do buzzer (`11`) para saída (`OUTPUT`).

Na função `loop()`, na condição de que o sensor está aberto e retornando `LOW` (depois do `else`), uso a variável `tempo` para receber o tempo decorrido até o instante, já em segundos. Por isso a divisão por 1000. A linha que faz isso é a `tempo = millis() / 1000`.

Com a estrutura de seleção `if(tempo - (tempo_fechar / 1000) > tempo_max)`, verifico se o tempo percorrido desde a abertura da porta é maior do que o tempo máximo determinado na variável `tempo_max`. A subtração `tempo - (tempo_fechar / 1000)` retornará o tempo percorrido entre o tempo registrado do último fechamento da porta e o tempo atual em segundos — por isso a divisão por 1000.

Caso essa condição seja verdadeira, o buzzer é desligado por 150 milissegundos e depois desligado pela mesma quantidade de tempo, fazendo com que ocorra um apito intermitente até que a porta da geladeira seja fechada.

Vamos então guardar a quantidade de aberturas da porta e os tempos dessas aberturas. Usarei a memória EEPROM do microcontrolador. No caso do ATMEGA328 existente no Arduino UNO, teremos 1KiB (kiloByte) disponível, o que é muito além do que precisamos, portanto, suficiente.

Vou alterar o código novamente:

```
#include <EEPROM.h>

int tempo = 0;
int tempo_max = 30;

int led = 13;
int sensor = 2;
int buzzer = 11;

int ctrl_abre = 0;
int ctrl_fecha = 0;
int contador = -1;
```

```
unsigned long tempo_abrir = 0;
unsigned long tempo_fechar = 0;

void setup() {
  Serial.begin(9600);
  pinMode(buzzer,OUTPUT);
  pinMode(led, OUTPUT);
  pinMode(sensor, INPUT);
  if(EEPROM.read(0) != 9) {
    EEPROM.write(0,9);
    EEPROM.write(1,0);
    EEPROM.write(2,0);
  }
}

void loop() {
  if (digitalRead(sensor) == HIGH) {
    digitalWrite(led, HIGH);
    if (ctrl_abre == 0) {
      tempo_fechar = millis();
      ctrl_abre = 1;
      ctrl_fecha = 0;
      Serial.print("Porta ficou aberta por ");
      Serial.print((tempo_fechar - tempo_abrir) / 1000);
      Serial.println(" segundos");
      contador++;
      EEPROM.write(1,contador);
      if(EEPROM.read(2) < (tempo_fechar - tempo_abrir) / 1000) {
        EEPROM.write(2,round((tempo_fechar - tempo_abrir) / 1000))
;
      }
    }
  } else {
    digitalWrite(led, LOW);
    tempo = millis() / 1000;
    if(tempo - (tempo_fechar / 1000) > tempo_max) {
      digitalWrite(buzzer,HIGH);
      delay(150);
      digitalWrite(buzzer,LOW);
      delay(150);
    }
    if (ctrl_fecha == 0) {
      tempo_abrir = millis();
      ctrl_fecha = 1;
      ctrl_abre = 0;
      tempo = 0;
    }
```

```
  }
  if(Serial.available() > 0) {
    char lido = toUpperCase(Serial.read());
    if(lido == 'Q') {
      Serial.println("Nesse ciclo a porta da geladeira: ");
      Serial.print("Foi aberta ");
      Serial.print(EEPROM.read(1));
      Serial.println(" vezes.");
      Serial.print("Ficou aberta pelo tempo maximo de ");
      Serial.print(EEPROM.read(2));
      Serial.println(" segundos.");
    }
    if(lido == 'R') {
      Serial.println("Valores zerados, novo ciclo iniciado.");
      contador = 0;
      EEPROM.write(1,0);
      EEPROM.write(2,0);
    }
  }
}
```

Veja que dessa vez as mudanças também não foram grandes. Incluí a linha `#include <EEPROM.h>` logo na primeira linha do código para importar a biblioteca EEPROM, que contém todos os métodos para facilitar o uso dessa memória.

Declarei uma variável nova, do tipo inteiro, com o nome de `contador`, inicializada com o valor `-1`. Dei esse valor porque ela está no mesmo trecho em que o programa passa por onde detectaria o fechamento da porta para inicializar as variáveis de tempo na primeira vez que é inicializado. Se a inicializasse com 0, a contagem seria incorreta, contando uma abertura e um fechamento de porta a mais do que o real.

Na função `setup()`, incluí a condição `if(EEPROM.read(0) != 9)`, que lê a posição 0 da memória EEPROM e verifica se nela há o número 9. O número 9 será nosso indicador de que esse programa já foi executado anteriormente e, dessa forma, há dados já gravados na EEPROM. Caso não seja o número 9 na posição 0 da EEPROM, usamos o comando `EEPROM.write(0,9)` para colocá-lo

lá.

Nas duas linhas seguintes, usamos os comandos `EEPROM.write(1,0)` e `EEPROM.write(2,0)`, que gravam o valor zero nas posições 1 e 2 da memória EEPROM. Essas duas posições serão utilizadas para guardar a quantidade de vezes que a porta da geladeira é aberta e fechada, e qual o tempo máximo em que permaneceu aberta, respectivamente.

Na função `loop()`, estão a maioria das mudanças. Incluí a linha `contador++` para incrementar cada vez que a porta é fechada e, logo em seguida, a linha `EEPROM.write(1,contador)` para guardar esse valor na posição 1 da memória EEPROM.

Em seguida, uso a estrutura de seleção `if(EEPROM.read(2) < (tempo_fechar - tempo_abrir) / 1000)` para verificar se o tempo guardado na posição 2 da memória EEPROM é menor do que o aferido no fechamento da porta. Caso seja, o tempo armazenado na memória precisa ser substituído pelo atual, já que estamos interessados no tempo máximo em que a porta ficou aberta no ciclo de operação atual. Caso contrário, nada é alterado. A linha para gravar o tempo na posição 2 da EEPROM é `EEPROM.write(2,round((tempo_fechar - tempo_abrir) / 1000))`.

E para conseguir ver esses dados gravados na EEPROM? Basta abrir o Serial Monitor e enviar a letra `Q` (tanto faz maiúscula como minúscula). Isso funciona graças à verificação `if(Serial.available() > 0)` que foi colocada no final da função `loop()`.

O comando `Serial.available()` retorna a quantidade de bytes disponíveis. Então, caso haja algo enviado pelo usuário, ele executará as linhas de comandos dentro dela.

Com a linha `char lido = toUpperCase(Serial.read())`, criamos uma variável chamada `lido` do tipo `char`, que armazenará o que for lido do Monitor Serial convertido para letra maiúscula. Dessa forma, se o usuário digitar `q` (minúsculo), ou `Q` (maiúsculo), ele entrará na condição, por exemplo.

Com `if(lido == 'Q')`, comparamos o que foi lido do Monitor Serial com a letra `Q` maiúscula. Dentro dessa condição, mostramos a quantidade de aberturas e fechamentos da porta da geladeira com a mensagem `"Nesse ciclo a porta da geladeira: Foi aberta"`. Também com a leitura da posição 1 da memória EEPROM feita com o comando `EEPROM.read(1)`, `" vezes"`. Continua com `"Ficou aberta pelo tempo Maximo de "`, com a leitura da posição 2 da memória EEPROM feita com o comando `EEPROM.read(2)`, `" segundos"`.

Para testar, compile e faça upload do Sketch para seu Arduino. Depois, com o Monitor Serial aberto, abra e feche o sensor algumas vezes e, em seguida, envie a letra `Q` ou `q`:

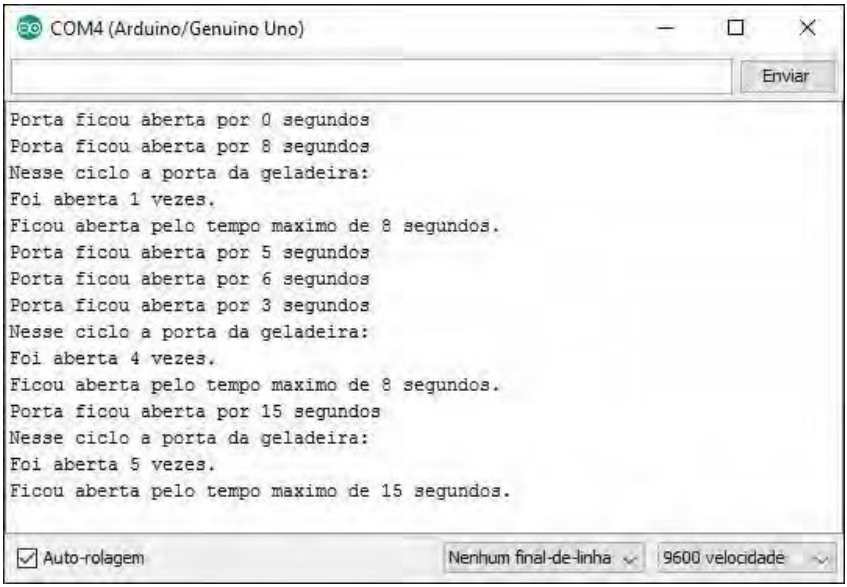


Figura 8.7: O resultado da contabilidade de aberturas e fechamentos da porta no Monitor Serial

Ainda há uma última estrutura de seleção `if(lido == 'R')`, que faz com que os valores armazenados nas posições 1 e 2 da EEPROM e também a variável contador sejam zerados caso queira iniciar um novo ciclo de checagem da porta da geladeira. Para que isso aconteça, basta enviar a letra `R` ou `r` via Monitor Serial.

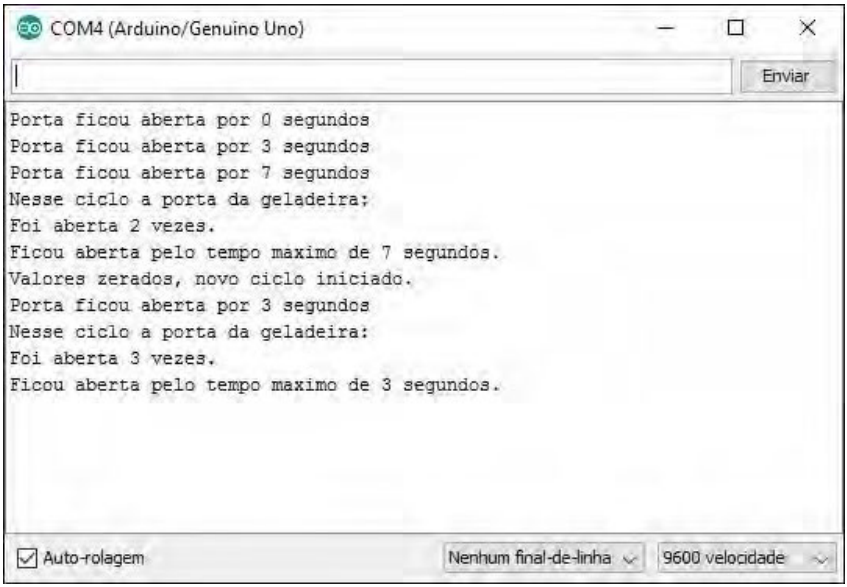


Figura 8.8: Zerando os valores armazenados na EEPROM

Agora é só arrumar uma fonte para ligar seu Arduino na tomada e instalar o sensor na porta da sua geladeira. Eu usei uma fonte de celular com 5V 1A e um fio USB com 2 metros para colocá-lo no lugar. Para ler os dados, basta desconectar o cabo USB da fonte e colocar em um computador.

Você pode ver o vídeo desse projeto instalado em uma geladeira e funcionando em https://youtu.be/KHYyWZxV5lk.

8.3 DESAFIO

Como você deve ter percebido, não preparei nenhum case para nenhum projeto. Então, acho que esse pode ser um dos desafios para esse projeto: arrumar um caixinha que você possa colar na

geladeira, no móvel ou na parede que você tiver perto dela.

Para colocar na caixinha, você precisará de uma prot-o-board menor, ou construir sua própria placa de circuito impresso. Outra ideia interessante é usar uma conexão sem fio para comunicar-se com o Arduino, como por exemplo, usando um módulo Bluetooth.

No próximo projeto, você poderá guardar os caminhos que faz pela cidade ou em viagens através de um rastreador GPS, que guarda suas posições geográficas em um cartão SD.

CAPÍTULO 9

PROJETO Nº 08 — CONSTRUINDO UM RASTREADOR GPS OFFLINE

Hoje em dia, talvez haja milhares de marcas e modelos de rastreadores GPS no mercado, e alguns muito sofisticados. Mas que tal construirmos nosso próprio rastreador offline? Ou seja, sem envio dos dados via linha telefônica móvel. A ideia será ler dados de um GPS e gravá-los em um cartão SD.

Você poderá passear onde desejar carregando o rastreador, e depois lê-lo em um computador qualquer. Com apenas uma conexão à internet, poderá ver o caminho que você fez. Como é possível deixar esse projeto muito pequeno e leve, você também pode tentar rastrear animais de estimação e saber por onde anda seu gatinho quando ele sai durante a noite.

9.1 MATERIAIS UTILIZADOS NESSE PROJETO

- 1 x Arduino UNO
- 1 x Módulo GPS
- 1 x Módulo cartão SD

- 1 x Cartão SD
- Fios diversos

9.2 DESENVOLVENDO O PROJETO

Primeiro, precisaremos de um GPS. Mas não esses GPS automotivos, precisamos de um módulo GPS.

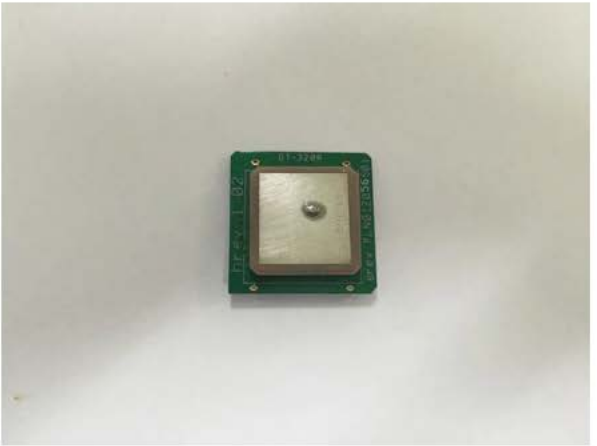


Figura 9.1: Vista superior de um módulo GPS

Quase todos os módulos GPS funcionam do mesmo jeito, e o que vou utilizar é o mais comum de encontrar. Para que você faça esse projeto, basta preocupar-se com que seu módulo tenha saída serial e possa ser alimentado com os 5V do Arduino.

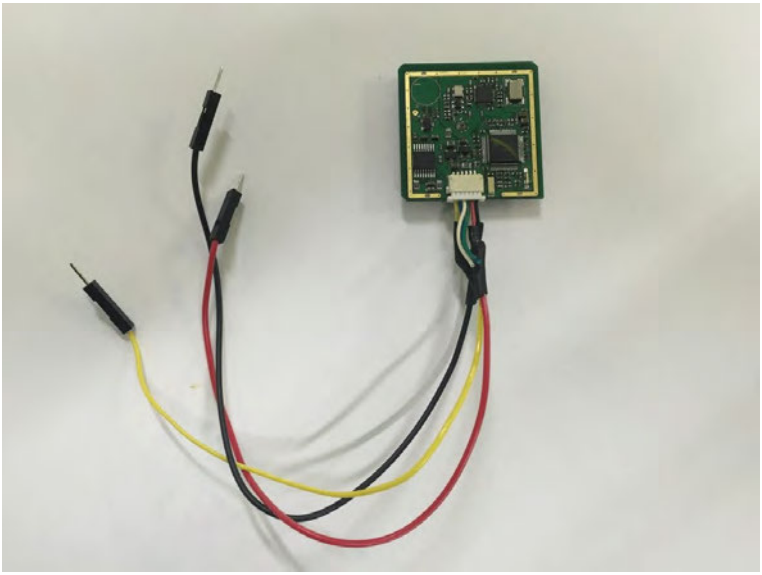


Figura 9.2: Vista do circuito de um módulo GPS junto com o conector e fios

No caso do módulo que estou usando, tive de recorrer ao datasheet para saber quais pinos correspondiam a qual funcionalidade. Para deixar isso bem evidente, estendi os fios do conector seguindo o padrão de cores do datasheet: vermelho para 5V, preto para GND e amarelo para transmissor serial (TX).

Módulos GPS são receptores de sinal GPS e que fornecem, via comunicação serial, os dados obtidos em forma de linhas de texto. Bom, vamos ligar tudo e depois entender exatamente como os dados de geolocalização são lidos.

Ligue o pino de alimentação 5V do seu GPS ao pino 5V do Arduino, o pino GND do GPS em um dos pinos GND do Arduino e o pino de comunicação serial (TX) ao pino digital 2 do Arduino:

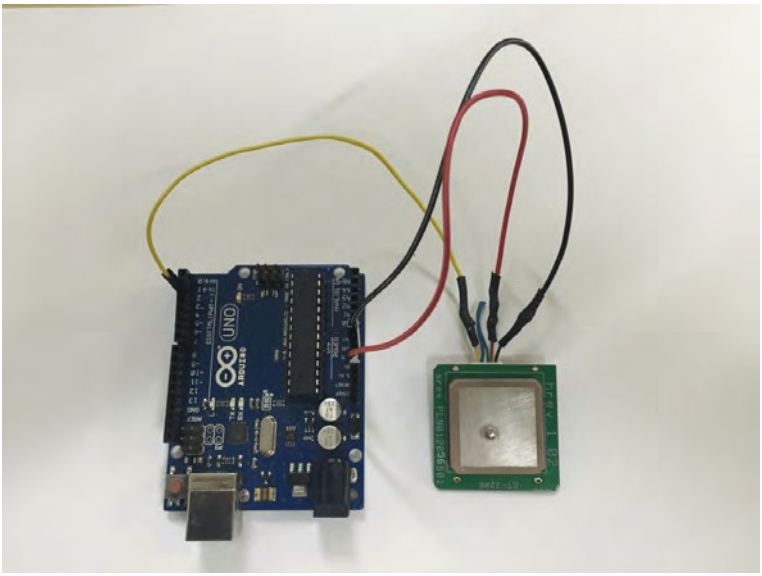


Figura 9.3: Módulo GPS ligado ao Arduino

Agora vamos ao código para ler os dados que o GPS envia:

```
#include <SoftwareSerial.h>

SoftwareSerial gps(2,3);

void setup() {
  gps.begin(4800);
  Serial.begin(9600);
}

void loop() {
  char lido = 0;
  String resposta = "";
  while((lido = gps.read()) != 10) {
    if(lido > 0) {
      resposta += lido;
    }
  }
  if(!resposta.equals("")) {
    Serial.println(resposta);
  }
}
```

A primeira coisa a fazer é importar a biblioteca `SoftwareSerial`, que nos permitirá criar uma comunicação serial com dois pinos quaisquer do Arduino. Isso é necessário porque não podemos usar os pinos digitais 0 e 1, que são, respectivamente, RX e TX nativos do Arduino. Se fizermos isso, comprometeremos a capacidade de vermos os dados no Monitor Serial.

Com a linha `SoftwareSerial gps(2,3)`, indicamos que os pinos digitais 2 e 3 serão utilizados para, respectivamente, receber (RX) e transmitir (TX) dados seriais através da porta serial virtual, que será criada e atenderá pelo nome de `gps`. Portanto, ligaremos o pino TX (transmissor) do GPS ao RX (receptor) da porta serial virtual que estamos criando.

Na função `setup()`, usamos `Serial.begin(9600)` para inicializar a porta serial física do Arduino, nos pinos 0 (RX) e 1 (TX), com a taxa de transferência de 9.600bps, como já fizemos em quase todos os outros projetos. Com a linha `gps.begin(4800)`, inicializamos a porta serial virtual criada nos pinos 2 (RX) e 3 (TX), com a taxa de transferência de 4.800bps, requerida para esse módulo GPS.

Na função `loop()`, criei duas variáveis, sendo elas `lido`, que é do tipo `char` e receberá cada caractere enviado pelo GPS; e a `resposta`, que é do tipo `String` e receberá a string final contendo todos os dados da geolocalização fornecidos pelo GPS. O fato é que o GPS envia os dados através de linhas de texto, usando como separador a vírgula e contendo todos os dados de geolocalização. Veremos isso em detalhes logo adiante.

Com o laço de repetição `while((lido = gps.read()) != 10)`, fazemos com que os caracteres sejam lidos até que o caractere ASCII número 10 seja encontrado, que é o indicador de final de linha. Com a seleção `if(lido > 0)`, apenas temos certeza de que o caractere é válido para ser armazenado.

Em `resposta += lido` , acumulamos os caracteres para formar a linha de texto contendo todos os dados na variável resposta. Finalmente usamos a estrutura de seleção `if(!resposta.equals(""))` para nos certificarmos de que uma linha de resposta foi formada (ou seja, diferente de vazio) e mostramos o que foi lido no Monitor Serial.

Para ver os dados gerados pelo GPS, carregue o programa em seu Arduino e abra o Monitor Serial:

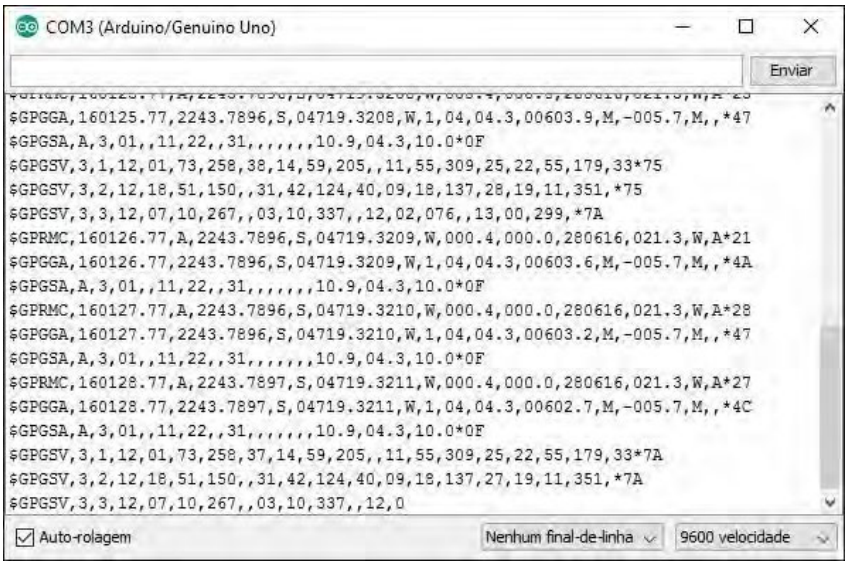


Figura 9.4: Os dados recebidos pelo GPS

Agora vamos entender tudo isso.

As linhas que você vê no Monitor Serial correspondem aos dados obtidos pelos satélites. Eles são mostrados no formato NMEA (*National Marine Electronics Association*) que são um conjunto de especificações de como os GPS devem ser construídos e como devem apresentar os dados.

O padrão NMEA diz que cada linha deve começar com cifrão

($), os próximos 5 caracteres indicam a origem da mensagem, sendo os dois primeiros para a origem (GP para GPS e GL para GLONASS) e os três seguintes para o tipo de mensagem (GSA, GSV, RMC e GGA). Todos os campos são separados por vírgulas, e cada tipo de mensagem tem seus campos específicos, que vou detalhar a seguir.

GSA — Dados gerais de satélite

Linha exemplo:

```
$GPGSA,A,3,26,21,,,09,17,,,,,,,10.8,02.1,10.6*07
```

Onde:

Campo	Exemplo	Descrição
1	A	Modo de operação, sendo:
		A – automático
		M – manual
2	3	Tipo de correção, sendo:
		1 – não disponível
		2 – correção 2D
		3 – correção 3D
3	26,21,,,09,17,,,,,,	Números dos satélites utilizados, de 1 até 32, máximo 12 por transmissão
4	10.8	Diluição da precisão da posição de 00.0 até 99.9
5	02.1	Diluição horizontal da precisão da posição de 00.0 até 99.9
6	10.6	Diluição vertical da precisão da posição de 00.0 até 99.9
7	07	Checksum. Número hexadecimal iniciado por * (asterisco) que indica um OR-exclusivo lógico entre todos os caracteres retornados entre o caractere $ (cifrão) e * (asterisco) da String.

GSV — Dados detalhados de satélite

Linha exemplo:

```
$GPGSV,2,1,08,26,50,016,40,09,50,173,39,21,43,316,38,17,41,144,42*
7C
```

Onde:

Campo	Exemplo	Descrição
1	2	Número total de mensagens GSV a serem transmitidas
2	1	Número da mensagem GSV atual
3	08	Total de satélites em visada de 0 até 12
4	26	Número do satélite, sendo:
		01 até 32 – GPS
		33 até 64 – SBAS
5	50	Elevação do satélite de 00 até 99 graus
6	016	Azimute do satélite de 00 até 359 graus
7	40	Potência de 00 até 99 dB, sendo *null* quando não recebe dados
8	7C	Checksum. Número hexadecimal iniciado por * (asterisco) que indica um OR-exclusivo lógico entre todos os caracteres retornados entre o caractere *$* (cifrão) e * (asterisco) da String.

RMC — Mínimo de dados recomendados para GPS

Linha exemplo:

```
$GPRMC,104549.04,A,2447.2038,N,12100.4990,E,016.0,221.0,250304,003
.3,W,A*22
```

Onde:

Campo	Exemplo	Descrição
1	104549.04	Tempo universal no formato *hhmmss.ss* de 000000.00 até 235959.99
2	A	Status, sendo:

		A – posição válida
		V – alerta do receptor de navegação
3	2447.2038	Latitude no formato *ddmm.mmmm* com zeros à esquerda
4	N	Indicador de hemisfério latitudinal, N para Norte e S para Sul
5	12100.4990	Longitude no formato *ddmm.mmmm* com zeros à esquerda
6	E	Indicador de hemisfério longitudinal, E para Leste e W para Oeste
7	016.0	Velocidade em knots (nós) de 000.0 até 999.99
8	221.0	Direção em graus de 000.0 até 359.9
9	250304	Data no formato *ddmmaa*
10	003.3	Variação magnética de 000.0 até 180.00 graus
11	W	Direção da variação magnética, sendo:
		E para leste
		W para oeste
12	A	Indicador de modo de operação, sendo:
		N – dados inválidos
		A – autônomo
		D – diferencial
		E - estimado
13	22	Checksum. Número hexadecimal iniciado por * (asterisco) que indica um OR-exclusivo lógico entre todos os caracteres retornados entre o caractere $ (cifrão) e * (asterisco) da String.

GGA — Informações de correção

Linha exemplo:

```
$GPGGA,104549.04,2447.2038,N,12100.4990,E,1,06,01.7,00078.8,M,0016
.3,M,,*5C
```

Onde:

Campo	Exemplo	Descrição
1	104549.04	Tempo universal de 000000.00 até 235959.99
2	2447.2038	Latitude no formato *ddmm.mmmm* com zeros à esquerda
3	N	Indicador de hemisfério latitudinal, N para Norte e S para Sul
4	12100.4990	Longitude no formato *ddmm.mmmm* com zeros à esquerda
5	E	Indicador de hemisfério longitudinal, E para Leste e W para Oeste
6	1	Indicador de correção para a qualidade da posição, sendo:
		0 – posição inválida
		1 – posição válida, modo SPS
		2 – posição válida, modo GPS diferencial
7	06	Número de satélites em uso de 0 até 12
8	01.7	Diluição horizontal da precisão de 0.0 até 99.9
9	00078.8	Altitude em relação ao nível do mar de -9999.9 até 17999.9
10	0016.3	Altitude geoidal de -999.9 até 9999.9
11	M (ASCII 77)	Tempo desde a última transmissão válida em segundos, sendo *null* caso do não uso do Posicionamento Global Diferencial (DGPS)
12		Identificação da estação diferencial de 0000 até 1023
13	5C	Checksum. Número hexadecimal iniciado por * (asterisco) que indica um OR-exclusivo lógico entre todos os caracteres retornados entre o caractere *$* (cifrão) e * (asterisco) da String.

Sabendo disso, podemos mudar nosso programa para ler apenas as strings do tipo RMC, que nos dará a latitude e longitude e outras informações básicas. Também podemos ler apenas as strings válidas, ou seja, com o campo nº 2 constando a letra A.

Com o código a seguir, o LED do pino digital 13 ficará piscando até que uma string RMC válida seja lida. Enquanto forem lidas strings válidas, o LED ficará aceso constantemente:

```
#include <SoftwareSerial.h>
```

```
SoftwareSerial gps(2,3);
int led = 13;
int led_status = LOW;

void setup() {
  gps.begin(4800);
  Serial.begin(9600);
  pinMode(led,OUTPUT);
}

void loop() {
  char lido = 0;
  String resposta = "";
  while((lido = gps.read()) != 10) {
    if(lido > 0) {
      resposta += lido;
    }
  }
  if(!resposta.equals("")) {
    if(resposta.indexOf("RMC") > 0) {
      if(resposta.charAt(17) == 'A') {
        Serial.println(resposta);
        led_status = HIGH;
      } else {
        if(led_status == LOW) {
          led_status = HIGH;
        } else {
          led_status = LOW;
        }
      }
    }
  }
  digitalWrite(led,led_status);
}
```

Esse é uma adaptação do código anterior, vamos nos concentrar apenas no que mudou. Declarei a variável `led` para indicar em qual pino digital o LED estará ligado. Também criei uma variável chamada `led_status`, que indicará qual é o valor enviado ao pino para ligar ou desligar o LED.

Na função `setup()`, ajustei o pino digital do LED como saída, usando a linha `pinMode(led,OUTPUT)`. O restante foi mudado dentro da função `loop()`, na estrutura de seleção

`if(!resposta.equals("")))` , ou seja, todo o resto só acontecerá se uma linha com dados for lida.

Com a linha `if(resposta.indexOf("RMC") > 0)` , uso o método `indexOf(string)` que retorna a posição onde a string procurada foi encontrada; caso contrário, retorna zero. Ou seja, se esse método retornar um valor maior do que zero, quer dizer que RMC foi encontrado na string.

Em seguida, com a linha `if(resposta.charAt(18) == 'A')` , verificamos se o caractere da 18ª posição da string resposta é igual à letra A. O caractere da 18ª posição é o campo nº 2 da mensagem tipo RMC, que indica o status do GPS, sendo que A indica que a posição emitida é válida.

Sendo a posição geográfica válida a enviamos para o Monitor Serial e fazemos com que o LED (`led_status`) fique aceso o tempo todo. Caso contrário o LED piscará, uma vez que esteja com led_status `LOW` para apagá-lo, no próximo loop `led_status` será `HIGH` para acendê-lo.

Já caminhando para a gravação no cartão SD, no próximo código, vamos separar a data, hora, latitude, longitude e velocidade em variáveis. No caso, vou usar um vetor para facilitar as coisas. Logo em seguida, vamos ligar o módulo de cartões SD e gravar o arquivo com esses dados no cartão.

```
#include <SoftwareSerial.h>

SoftwareSerial gps(2, 3);
int led = 13;
int led_status = LOW;
String campo[13];

void setup() {
  gps.begin(4800);
  Serial.begin(9600);
  pinMode(led, OUTPUT);
}
```

```
void loop() {
  char lido = 0;
  String resposta = "";
  while ((lido = gps.read()) != 10) {
    if (lido > 0) {
      resposta += lido;
    }
  }
  if (!resposta.equals("")) {
    if (resposta.indexOf("RMC") > 0) {
      if (resposta.charAt(17) == 'A') {
        decodeResposta(resposta);
        Serial.println(resposta);
        for(int i=0;i<13;i++) Serial.println(campo[i]);
        led_status = HIGH;
      } else {
        if (led_status == LOW) {
          led_status = HIGH;
        } else {
          led_status = LOW;
        }
      }
    }
  }
  digitalWrite(led, led_status);
}

void decodeResposta(String r) {
  int ant = 0;
  int pos = 0;
  for (int i = 0; i < 13; i++) {
    pos = r.indexOf(",", ant + 1);
    campo[i] = r.substring(ant, pos);
    ant = pos + 1;
  }
}
```

Vamos nos ater apenas ao que foi incrementado: Criei um vetor chamado `campo` com 13 posições do tipo `String`. Cada posição guardará um campo da mensagem do tipo NMEA RMC. No final do programa criei, uma função chamada `decodeResposta`, que recebe uma `String` que ficará armazenada na variável `r` para uso nesta função.

A ideia é encontrar as vírgulas, que são os separadores de campos, e guardar em cada posição do vetor um recorte da string NMEA correspondente a cada campo. Começamos com a variável `ant` (que indica a vírgula anterior) com zero, ou seja, no início da string. A variável `pos` (que indica a vírgula posterior) também começa com zero e será determinada no loop `for`.

Como a mensagem do tipo RMC tem 13 campos, nosso `for` vai até 13 (lembre-se de que o vetor começa na posição 0), então usamos `for(int i = 0; i < 13; i++)` para criar esse loop.

Na sequência, usamos `pos = r.indexOf(",", ant + 1)` para determinar a posição da vírgula posterior, que será a posição da ocorrência da primeira vírgula depois da próxima posição além da vírgula anterior. Em `campo[i] = r.substring(ant, pos)`, recortamos os dados do campo, que está entre as vírgulas encontradas.

Para o próximo laço, a posição da vírgula anterior será a posição da vírgula posterior adicionado de um. E assim sucessivamente, 12 vezes, até que todos os campos estejam dentro do vetor campo.

Na função `loop()`, usei a função que acabamos de criar com `decodeResposta(resposta)`, em seguida mandei para o Monitor Serial a string que recebemos do GPS usando a linha `Serial.println(resposta)`. Depois, com o laço `for(int i=1;i<13;i++) Serial.println(campo[i])`, enviamos cada campo separadamente. Dessa forma, podemos conferir se está tudo certo.

Depois de enviar o programa para seu Arduino, abra o Monitor Serial e você deverá ver algo como a figura:

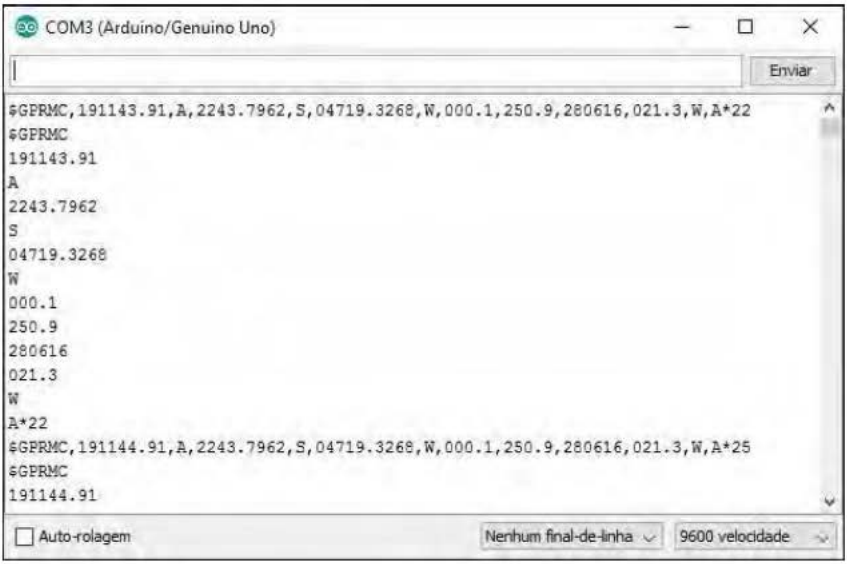


Figura 9.5: Monitor Serial com os campos RMC separados no vetor

Farei apenas mais uma alteração antes de partir para o cartão SD. Vamos formar os dados dos campos. A data ficará com o formato de correto e o mesmo para a hora.

Para a latitude e longitude, teremos de convertê-las de valor NMEA para decimal. Isso porque vamos mostrar esses dados no Google Maps e ele só interpreta coordenadas decimais. Você pode até fazer um teste.

Abra seu navegador de internet e digite a seguinte URL: http://maps.googleapis.com/maps/api/staticmap?center=latitude,longitude&size=500x500&zoom=15.

Para latitude e longitude, digite os números que o GPS está lhe dando. No meu caso, baseado na figura anterior, ficaria assim: http://maps.googleapis.com/maps/api/staticmap?center=2243.8,04719.3&size=500x500&zoom=15.

Veja o resultado:

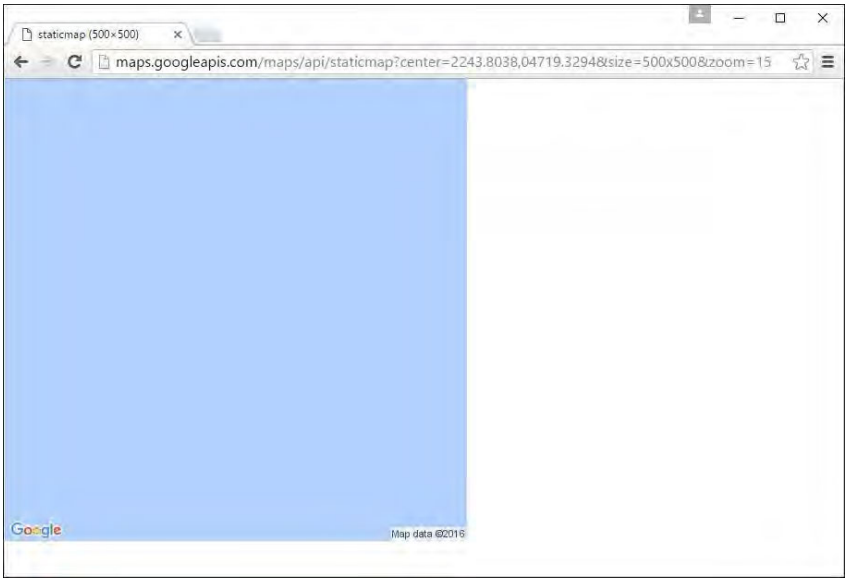


Figura 9.6: Google Maps mostrando uma posição qualquer no meio do oceano

A URL que formamos indica o endereço da API do Google Maps, e as variáveis `center` indicam a posição geográfica do centro do mapa. `size` é o tamanho em pixels que o mapa deve ter e o índice de zoom, que no nosso caso é 15.

Agora vamos converter essas latitude e longitude de NMEA para decimal. Para a latitude, pegue o número composto pelos dígitos dos centésimos e some ao número composto pelo restante dos dígitos divididos por 60. Veja:

Latitude NMEA (GPS) = 2243.8038

Latitude decimal = 22 + (43.8038/60) = 22 + 0.7300633 = 22.7300633

Para hemisfério latitudinal Sul, o valor deve ser negativo; caso

contrário, positivo.

Latitude decimal final = -22.7300633

Faça o mesmo para a longitude, mas usando os dígitos dos centésimos para formar o primeiro número:

Longitude NMEA (GPS) = 04719.3294 Longitude decimal = 47 + (19.3294/60) = 47 + 0.3221566 = 47.3221566

Para hemisfério longitudinal Oeste, o valor deve ser negativo; caso contrário, positivo.

Longitude Decimal final = -47.3221566

Agora troque os valores da latitude e da longitude na URL pelo os que acabamos de calcular: http://maps.googleapis.com/maps/api/staticmap?center=-22.7,-47.3&size=500x500&zoom=15.

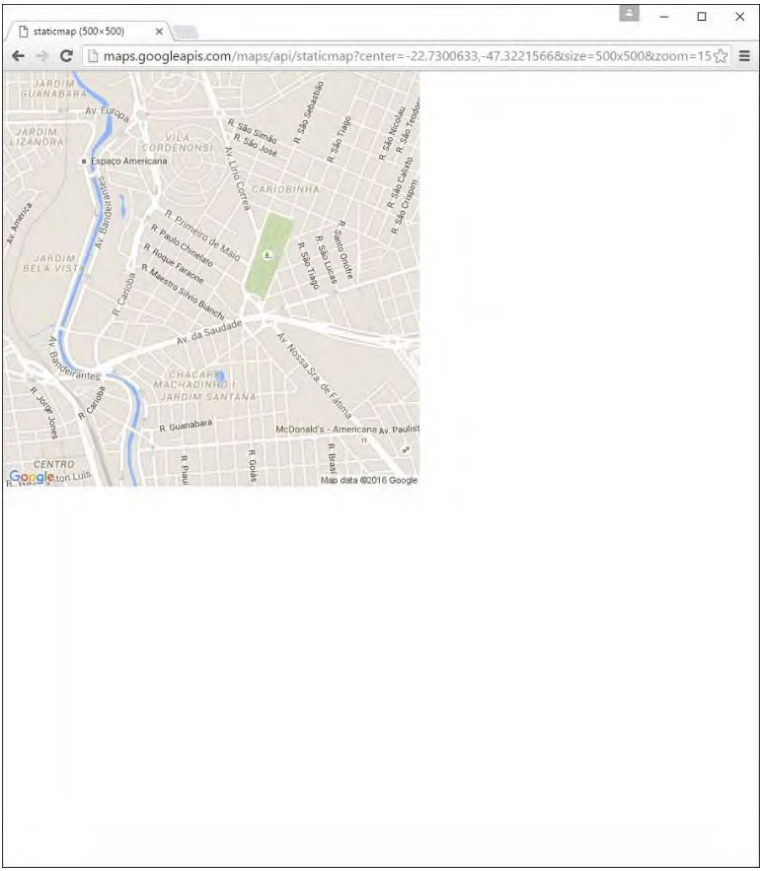


Figura 9.7: Mapa com as coordenadas corretas, você me achou!

Vamos traduzir tudo isso em código!

```
#include <SoftwareSerial.h>

SoftwareSerial gps(2, 3);
int led = 13;
int led_status = LOW;
String campo[13];

void setup() {
  gps.begin(4800);
  Serial.begin(9600);
  pinMode(led, OUTPUT);
}
```

```
void loop() {
  char lido = 0;
  String resposta = "";
  while ((lido = gps.read()) != 10) {
    if (lido > 0) {
      resposta += lido;
    }
  }
  if (!resposta.equals("")) {
    if (resposta.indexOf("RMC") > 0) {
      if (resposta.charAt(17) == 'A') {
        decodeResposta(resposta);
        Serial.println(resposta);
        Serial.println(formataHora(campo[1]));
        Serial.println(formataData(campo[9]));
        Serial.println(decNMEA(campo[3], campo[4]), 10);
        Serial.println(decNMEA(campo[5], campo[6]), 10);
        Serial.println(velocidadeKM(campo[7]));
        led_status = HIGH;
      } else {
        if (led_status == LOW) {
          led_status = HIGH;
        } else {
          led_status = LOW;
        }
      }
    }
  }
  digitalWrite(led, led_status);
}

void decodeResposta(String r) {
  int ant = 0;
  int pos = 0;
  for (int i = 0; i < 13; i++) {
    pos = r.indexOf(",", ant + 1);
    campo[i] = r.substring(ant, pos);
    ant = pos + 1;
  }
}

float velocidadeKM(String v) {
  return v.toFloat() * 1.852;
}

String formataHora(String hr) {
  return hr.substring(0, 2) + ":" + hr.substring(2, 4) + ":" + hr.
substring(4, 6);
```

```
}

String formataData(String dt) {
  return dt.substring(0, 2) + "/" + dt.substring(2, 4) + "/" + dt.
substring(4, 6);
}

double decNMEA(String l, String h) {
  int a = int(l.toFloat()) / 100;
  double b = (l.toFloat() - (a * 100)) / 60;
  if (h.equals("W") || h.equals("S")) {
    return (a + b)*-1;
  } else {
    return a + b;
  }
}
```

Na função `loop()`, usamos as linhas:

```
Serial.println(formataHora(campo[1]));
Serial.println(formataData(campo[9]));
Serial.println(decNMEA(campo[3], campo[4]), 10);
Serial.println(decNMEA(campo[5], campo[6]), 10);
Serial.println(velocidadeKM(campo[7]));
```

Para mostrar os resultados no Monitor Serial, chamamos as funções que criamos no final do programa. A função `formataHora` recebe uma string e retorna a mesma string, mas com um dois-pontos (`:`) a cada dois caracteres, para que fique no formato de hora.

A função `formataData` é igual à função `formataHora`, ou seja, recebe uma string e retorna a mesma string, mas com uma barra (`/`) a cada dois caracteres para que fique no formato de data.

A função `decNMEA` recebe duas string. A primeira é a latitude ou longitude, e a segunda o hemisfério. Ela realiza a conversão conforme o algoritmo explicado anteriormente, e retorna a latitude ou a longitude em decimal (`double`).

A função `velocidadeKM` recebe uma string com a velocidade em knots obtida com o GPS, e converte em um valor `float` em

quilômetros por hora.

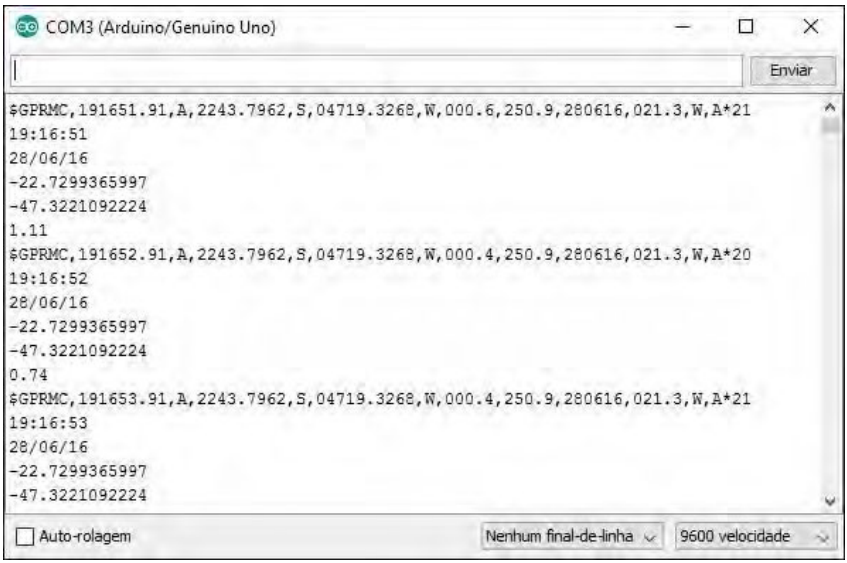


Figura 9.8: Os dados convertidos no Monitor Serial

Finalmente vamos ao módulo de cartões SD!

Você vai precisar também de um cartão SD, claro. Ele pode ter a capacidade que achar mais conveniente, pois isso apenas limitará a quantidade de dados que poderá ser armazenada. Fora isso, não influencia em mais nada no projeto.

Figura 9.9: O módulo para cartões SD e um cartão SD

Para ligá-lo ao Arduino, basta seguir a seguinte sequência:

- Ligue o terminal GND do módulo a um pino GND do Arduino;
- Ligue o terminal 3,3V do módulo ao pino 3,3V do Arduino;
- Ligue o terminal CS do módulo ao pino digital 4 do Arduino;
- Ligue o terminal MOSI do módulo ao pino digital 11 do Arduino;
- Ligue o terminal SCK do módulo ao pino digital 13 do Arduino;
- Ligue o terminal MISO do módulo ao pino digital 12 do Arduino;

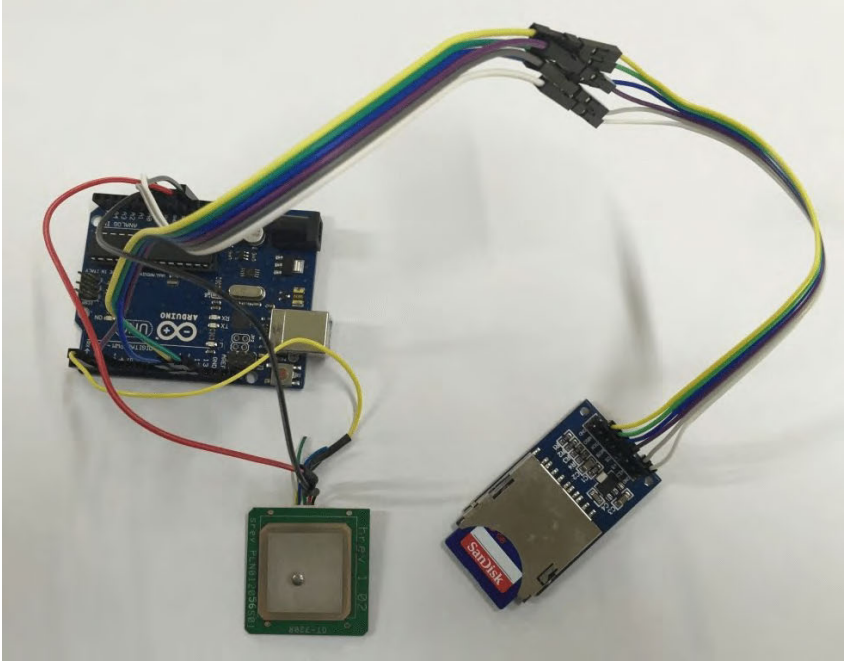


Figura 9.10: O módulo para cartões SD ligado ao Arduino

Antes de sairmos gravando os dados do GPS no cartão, vamos fazer um pequeno programa para testar o módulo e ver como a gravação funciona:

```
#include <SD.h>

void setup() {
  Serial.begin(9600);
  if(!SD.begin(4)) {
    Serial.println("Houve um erro em inicializar o modulo SD.");
    while(1);
  }
  Serial.println("Modulo SD inicializado com sucesso.");
}

void loop() {
  String texto = "Teste de gravação no cartão SD";
  File arquivo = SD.open("teste.txt", FILE_WRITE);
  if(arquivo) {
    arquivo.println(texto);
    arquivo.close();
```

```
    Serial.println("Texto gravado!");
    while(1);
  }
}
```

Usaremos a biblioteca `SD` já integrada ao IDE Arduino. Por isso, a primeira coisa que fazemos é importá-la com `#include <SD.h>`.

Na função `setup()`, iniciamos a comunicação serial com `Serial.begin(9600)`. Mostrar os dados no Monitor Serial será muito útil para termos um retorno visual do que está sendo feito.

Em seguida, usamos uma estrutura de seleção `if(!SD.begin(4))`. O comando `SD.begin(4)` inicia o módulo de cartões SD indicando que o pino digital 4 é o seletor de linha (CS — *chip select line*). Ele retornará verdadeiro se tiver sucesso, e falso caso não tenha. Portanto, checamos com o `if` e, caso a inicialização falhe, enviamos uma mensagem para a comunicação serial e usamos `while(1)` para interrompermos a execução do programa com um loop infinito. Já caso a inicialização tenha sucesso, mostramos mensagem no Monitor Serial com `Serial.println("Modulo SD inicializado com sucesso.")`.

Na função `loop()`, criamos uma variável texto, do tipo `String`, para conter a mensagem `"Teste de gravação no cartão SD"` que será gravada no arquivo a ser criado no cartão SD.

Criamos a variável `arquivo` como do tipo `File`. Ela recebe o ponteiro que faz referência ao arquivo aberto com `SD.open("teste.txt",FILE_WRITE)`. Nele `"teste.txt"` é o nome do arquivo, e `FILE_WRITE` o tipo de abertura, no caso, para gravação.

É preciso ter em mente que os arquivos apenas podem ter nomes no formato 8.3. Ou seja, 8 caracteres para o nome e 3 para a extensão.

Os modos de abertura de arquivos são `FILE_READ` para leitura, e `FILE_WRITE` para gravação. No caso do `FILE_WRITE`, caso o arquivo já exista, o novo conteúdo será colocado no final dele (*append*). Se o arquivo não puder ser aberto, a variável `arquivo` receberá o valor lógico `FALSE`, portanto usamos um `if(arquivo)` para verificar se o arquivo foi aberto corretamente.

Com o arquivo aberto corretamente, usamos `arquivo.println(texto)` para gravar o conteúdo da variável `texto` no arquivo. Além do método `println`, que fará a quebra de linha, também é possível usar `print` para que não ocorra a quebra de linha.

Antes que o arquivo seja fechado, a gravação não ocorrerá fisicamente no cartão. Logo, é importante lembrar-se de usar `arquivo.close()`. Você pode usá-lo sempre ao final, depois de enviar todo conteúdo. Mas caso se esqueça dele, os dados não serão gravados no cartão. O `while(1)` criará um loop infinito para que nada mais seja enviado ao cartão.

Depois de enviar o programa ao seu Arduino, abra o Monitor Serial e você deverá ter um resultado como o da figura a seguir:

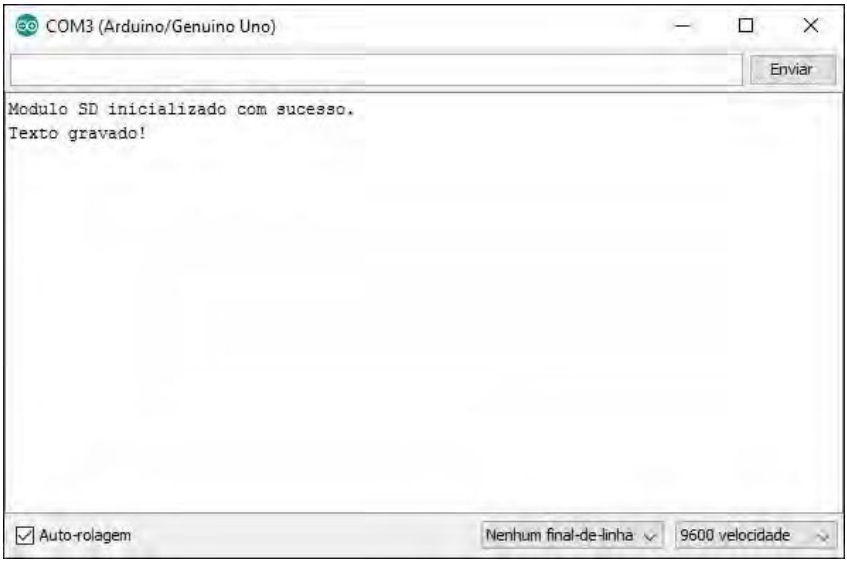


Figura 9.11: Mensagens confirmando a gravação da mensagem no cartão SD

As duas possibilidades de erro podem ser o módulo não estar funcionando, ou o arquivo não poder ser aberto no cartão (falta do cartão ou proteção à gravação):

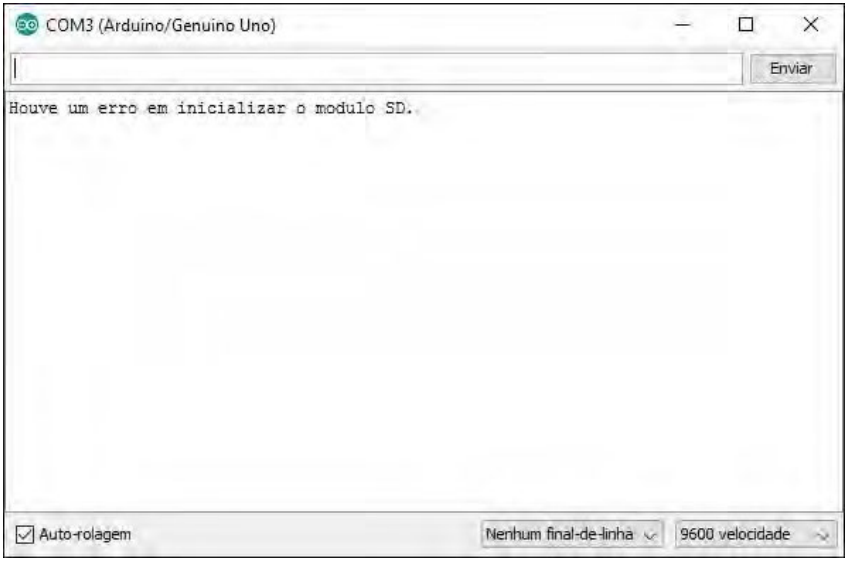


Figura 9.12: Mensagem de erro por causa de problemas no módulo

Bom, agora que sabemos tudo o que precisamos: obter os dados do GPS e gravar arquivos no cartão SD. Então, é hora de juntar tudo!

Vou usar a última versão do programa do GPS e juntá-la com esse programa de gravação no cartão SD.

```
#include <SD.h>
#include <SoftwareSerial.h>

SoftwareSerial gps(2, 3);
int led = 7;
int led_status = LOW;
String campo[13];

void setup() {
  gps.begin(4800);
  Serial.begin(9600);
  pinMode(led, OUTPUT);
  if(!SD.begin(4)) {
    Serial.println("Houve um erro em inicializar o modulo SD.");
    while(1);
  }
}

void loop() {
  File arquivo = SD.open("log_gps.txt", FILE_WRITE);
  char lido = 0;
  String resposta = "";
  while ((lido = gps.read()) != 10) {
    if (lido > 0) {
      resposta += lido;
    }
  }
  if (!resposta.equals("")) {
    if (resposta.indexOf("RMC") > 0) {
      if (resposta.charAt(17) == 'A') {
        decodeResposta(resposta);
        Serial.println(resposta);
        if(arquivo) {
          arquivo.print(decNMEA(campo[3], campo[4]), 10);
          arquivo.print(",");
          arquivo.print(decNMEA(campo[5], campo[6]), 10);
          arquivo.print("|");
          arquivo.close();
          Serial.println("Texto gravado!");
```

```
        }
        led_status = HIGH;
      } else {
        if (led_status == LOW) {
          led_status = HIGH;
        } else {
          led_status = LOW;
        }
      }
    }
  }
  digitalWrite(led, led_status);
}

void decodeResposta(String r) {
  int ant = 0;
  int pos = 0;
  for (int i = 0; i < 13; i++) {
    pos = r.indexOf(",", ant + 1);
    campo[i] = r.substring(ant, pos);
    ant = pos + 1;
  }
}

float velocidadeKM(String v) {
  return v.toFloat() * 1.852;
}

String formataHora(String hr) {
  return hr.substring(0, 2) + ":" + hr.substring(2, 4) + ":" + hr.
substring(4, 6);
}

String formataData(String dt) {
  return dt.substring(0, 2) + "/" + dt.substring(2, 4) + "/" + dt.
substring(4, 6);
}

double decNMEA(String l, String h) {
  int a = int(l.toFloat()) / 100;
  double b = (l.toFloat() - (a * 100)) / 60;
  if (h.equals("W") || h.equals("S")) {
    return (a + b)*-1;
  } else {
    return a + b;
  }
}
```

As únicas mudanças são: o número do pino digital do LED, que tirei do 13 e coloquei no 7, já que o 13 é usado para o módulo de cartões SD. Fiz todas as inicializações para usar o módulo de cartões SD, inclusive a importação da biblioteca. Finalmente, em vez de enviar os dados geográficos do GPS para o Monitor Serial, estou mandando para o cartão SD.

Não estou gravando a data, hora e velocidade, já que esses dados não serão necessários para mostrar no mapa. Mas você pode armazená-los sem maiores problemas seguindo o mesmo esquema. Também estou gravando a latitude e longitude no formato pronto para gerar a URL do Google Maps, como mostrei anteriormente.

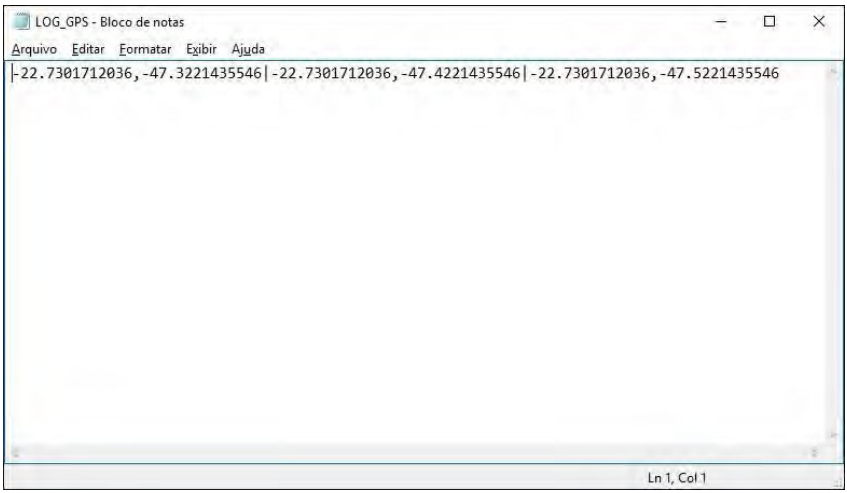


Figura 9.13: Latitude e longitude gravadas no arquivo log_gps.txt do cartão SD.

Para exibir esses pontos no Google Maps, basta montar a URL da seguinte forma: http://maps.googleapis.com/maps/api/staticmap?size=300x300&path=conteudo_do_arquivo_log_gps.txt.

O `conteudo_do_arquivo_log_gps.txt` é a cópia dos dados do arquivo do cartão SD. Se considerarmos os dados do arquivo

mostrado na figura anterior, a URL ficará assim: http://maps.googleapis.com/maps/api/staticmap?size=300x300&path=-22.7,-47.3|-22.7 36,-47.4|-22.7,-47.5.

Você verá no mapa uma linha indicando o caminho que essas posições geográficas indicam:

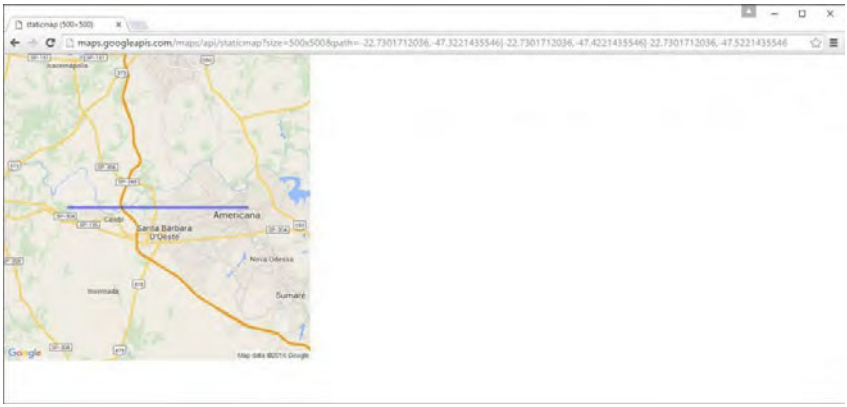


Figura 9.14: Vista do mapa gerado com a URL anterior no navegador de internet

O único cuidado que você precisa ter é que uma URL pode ter, no máximo, 2048 caracteres. Fato este que limita a quantidade de posições que podem ser mostradas. Além disso, é preciso criar uma página específica usando a API do Google Maps.

> Apensar de que é possível apenas ver o LED do GPS piscando, indicando seu funcionamento, um vídeo desse projeto em funcionamento pode ser visto em https://youtu.be/mb2zIKeh-q0.

9.3 DESAFIO

Os desafios são tantos que pode até ser que você pense em alguma coisa que eu jamais pensaria, e vice-versa. Primeiro, pode colocar tudo dentro de uma caixa que seja impermeável, se você for carregar o rastreador na bicicleta, por exemplo. Se for colocar no carro, você pode fazer com que o Arduino seja alimentado na tomada 12 volts do veículo.

É possível ir muito além trocando o cartão SD por um módulo GPRS, e enviar isso diretamente para a internet, de forma online. Assim, começar a criar seu próprio rastreador GPS profissional.

Pode ainda colocar alguns LEDs indicativos para saber se o GPS está retornando posições válidas ou não, se o cartão está inserido ou não, e até se há qualquer outro erro ou status que você queira acompanhar. Para não gravar muitos dados e logo encher o cartão, você pode escolher gravar apenas depois de algum tempo, alguma distância ou alguma mudança de direção predeterminada.

No próximo projeto, construiremos um brinquedo, porque ninguém é de ferro e precisamos de um pouco de diversão com os irmãos, irmãs ou filhos pequenos. Vamos construir uma batata quente!

Capítulo 10

PROJETO Nº 09 — BATATA QUENTE, QUENTE... QUENTE... QUEIMOU!

Os melhores projetos de robótica envolvendo Arduino talvez sejam os brinquedos ou jogos. Principalmente se você tem em mente o uso deles em sala de aula ou com crianças. É garantia de sucesso.

Há, literalmente, uma infinidade de projetos que podem ser feitos para a diversão de todos. Esse da batata quente é um dos que mais divertem meus filhos. Em vez de ficarmos passando uma batata de verdade ou uma bolinha e cantando, um Arduino para gerar um tempo aleatório e um buzzer para gerar barulho criam a tensão que essa brincadeira requer.

O brinquedo vai apitando durante uma quantidade de vezes sorteada aleatoriamente e, no final, emite um silvo longo indicando que "queimou" quem estava com a "batata" na mão.

10.1 MATERIAIS UTILIZADOS NESSE PROJETO

- 1 x Arduino UNO R3
- 1 x Buzzer
- 1 x Suporte para baterias 9V com plug para o Arduino

- 1 x Bateria 9V

10.2 DESENVOLVENDO O PROJETO

Bom, primeiro você precisará ter um Arduino UNO:

Figura 10.1: O Arduino UNO

O buzzer é um componente eletrônico composto por uma membrana vibratória em um invólucro normalmente de plástico. Ao excitá-lo eletricamente, a membrana vibra e emite um ruído.

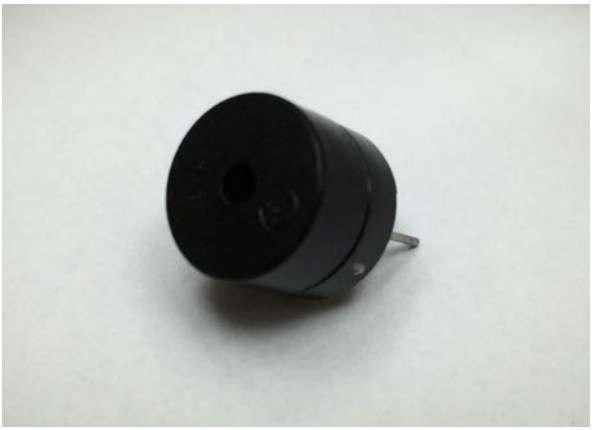


Figura 10.2: Um buzzer, que em sua parte superior tem a indicação do pino positivo

Além do Arduino e do buzzer, as únicas coisas que você precisará serão um suporte para baterias 9V com plug para ligá-lo ao Arduino, e de uma bateria também.

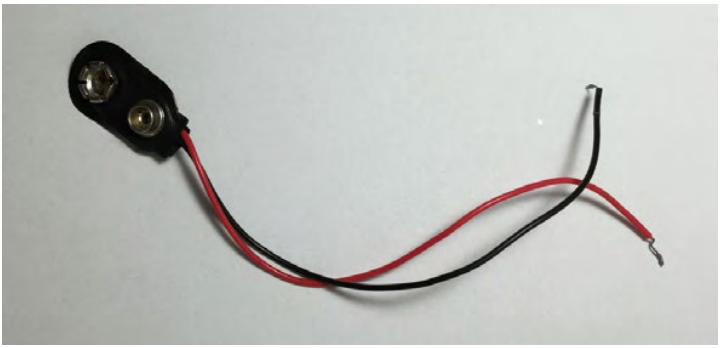


Figura 10.3: O plug para baterias de 9V

Para montar o suporte de baterias com o plug, basta soldar o fio positivo (vermelho) no pino central do plug, como na figura:

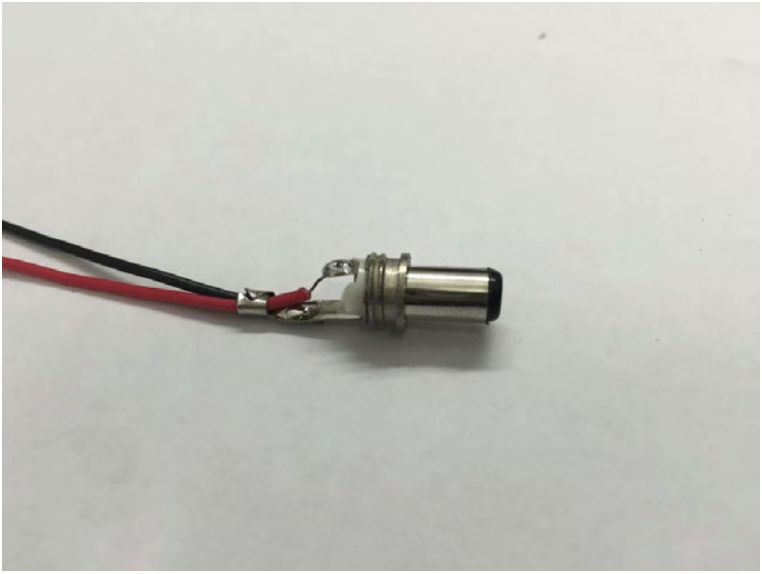


Figura 10.4: Soldando o fio positivo (vermelho) no pino central do plug e o negativo (preto) no pino externo

Lembre-se de deixar a capa já colocada nos fios para que seja afixada sobre o conector posteriormente, que montado, ficará como na figura a seguir:

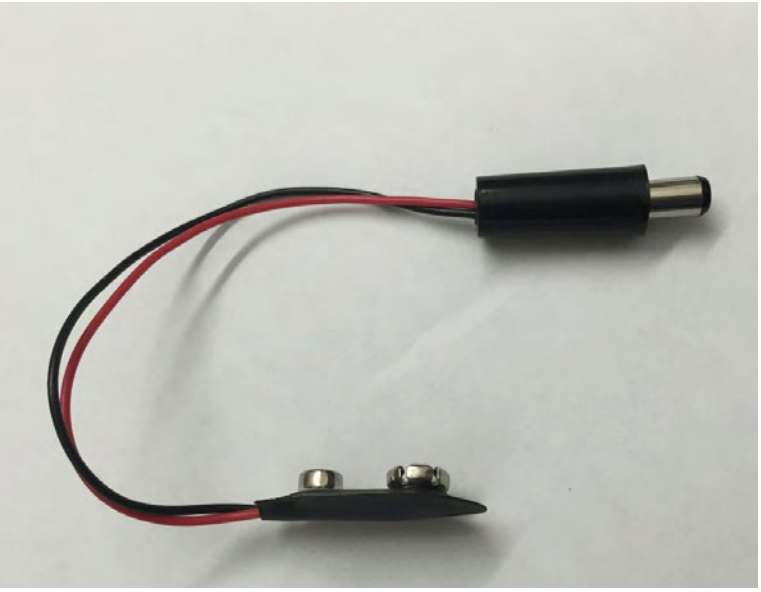


Figura 10.5: Clip de bateria com o conector montado

Para testar se está tudo OK, certificando-se de que o positivo da bateria esteja ligado ao pino central do conector, conecte uma bateria ao suporte e todo o conjunto a um Arduino:

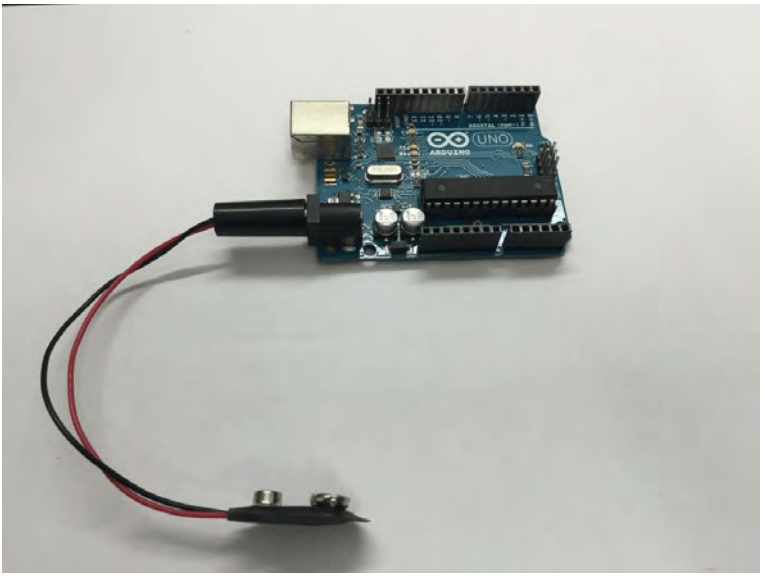


Figura 10.6: Conjunto com o suporte de bateria e conector ligado ao Arduino!

A montagem será simples, não usaremos fios extras nem qualquer outra coisa. Vamos aproveitar os próprios pinos do buzzer e ligar o pino positivo do buzzer ao pino digital 11 do Arduino, e o pino negativo do buzzer ao pino GND do Arduino, como na figura seguinte:

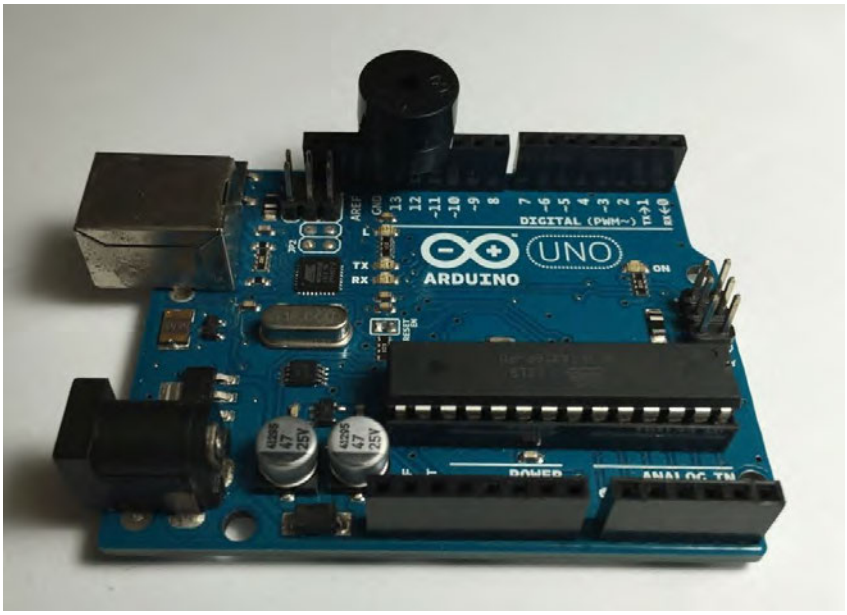


Figura 10.7: O buzzer conectado diretamente aos pinos do Arduino

Dessa forma, o projeto ficará compacto e poderá ser facilmente colocado em qualquer invólucro, como você verá mais adiante.

Vamos à programação:

```
int buzzer = 11;
int vezes = 0;

void setup() {
    pinMode(buzzer, OUTPUT);
    randomSeed(analogRead(A0));
    vezes = random(10, 100);
    for (int i = 0; i <= vezes; i++) {
        digitalWrite(buzzer, HIGH);
        delay(100);
        digitalWrite(buzzer, LOW);
        delay(100);
    }
    digitalWrite(buzzer, HIGH);
    delay(1000);
    digitalWrite(buzzer, LOW);
}
```

```
void loop() {
}
```

Usei duas variáveis do tipo inteiro. Uma é a chamada `buzzer`, para armazenar o pino digital em que o buzzer está conectado. Você pode alterar o seu valor caso conecte seu buzzer em outro pino qualquer. A outra, chamada `vezes`, é iniciada com zero e indicará a quantidade de vezes que o buzzer vai soar.

Coloquei todo o código na função `setup()` para que, após cada rodada, seja necessário pressionar o botão *reset* do Arduino. Isso faz com que seja possível controlar o início e término de cada rodada da brincadeira.

Como você já reparou, a programação é simples e pequena, mas vamos aos detalhes mesmo assim. Com o `pinMode(buzzer,OUTPUT)`, indicamos que o pino do buzzer (11) será de saída, ou seja, colocaremos tensão nele.

Em seguida, uso o comando `randomSeed(analogRead(A0))` para inicializar a semente de números pseudoaleatórios do Arduino. Como precisamos passar um número para que a função gere os números, passei o pino analógico A0 que está desconectado de qualquer coisa. Isso resultará em um ruído, aumentando a chance de os números gerados serem totalmente diferentes a cada vez que o programa começar a rodar.

Na linha seguinte, com `vezes = random(10, 100)`, geramos um número aleatório entre 10 e 100 e o armazenamos na variável vezes. Isso garantirá que o buzzer soará, no mínimo, 10 vezes e no máximo 100, permitindo que haja a troca da batata entre quem estiver brincando, mas sem ser curto nem longo demais, não deixando as pessoas entediadas.

A estrutura de repetição `for` fará com que o buzzer seja acionado durante 100 milissegundos, gerando um silvo breve e

agudo, e permanecendo desligado também por 100 milissegundos. Assim, cada laço de repetição vai durar 200 milissegundos.

Depois de cumpridos os laços de repetição, o buzzer será acionado por 1 segundo e desligado em seguida, indicando o final da rodada da brincadeira. Para recomeçar, basta pressionar o botão *reset* do Arduino.

Você encontrará um vídeo desse projeto em funcionamento em https://youtu.be/gwo2ChgszKM.

10.3 DESAFIO

As melhorias podem ser infinitas, mas deixo o desafio para esse projeto em usar seus conhecimentos artísticos e construir um invólucro que crie a batata, para que possa ser jogada entre os participantes da brincadeira. Também ficará bem divertido se, em vez de bips, você consiga realizar alguma melodia, talvez alguma cantiga de roda que ficará muito melhor se você não usar um buzzer, mas sim um autofalante.

Você também pode unir esse projeto àquele em que construímos nosso próprio Arduino para eliminar a placa completa. Isso deixará a eletrônica mais compacta e fácil de ser integrada ao invólucro que você criará.

Agora, se você achou esse projeto simples e fácil, veja o que vem a seguir: um robô tão fácil de construir e programar quanto a nossa batata quente! Isso mesmo, o robô aranha mais simples e fácil de construir do mundo, sem medo de errar. Preparado?

Vamos lá ver do que se trata!

Capítulo 11

PROJETO Nº 10 — O ROBÔ ARANHA MAIS SIMPLES DO MUNDO!

Não ache que estou exagerando em classificar esse projeto como sendo o robô aranha mais simples do mundo, porque, além de simples, é muito fácil de construir.

O fato é que você não precisará de nada além do que já deve dispor em casa. Ok, na casa de um maker!

Ele se movimentará como uma aranha. Mas, em vez de oito patas como o bichinho real tem, o nosso robô se apoiará em apenas quatro.

11.1 MATERIAIS UTILIZADOS NESSE PROJETO

- 1 x Arduino UNO
- 2 x Servos motores
- 3 x Clips nº 10
- 1 x Suporte para 4 pilhas
- 4 x Pilhas
- Fios diversos

11.2 DESENVOLVENDO O PROJETO

Como todo projeto, você precisará de um Arduino Uno:

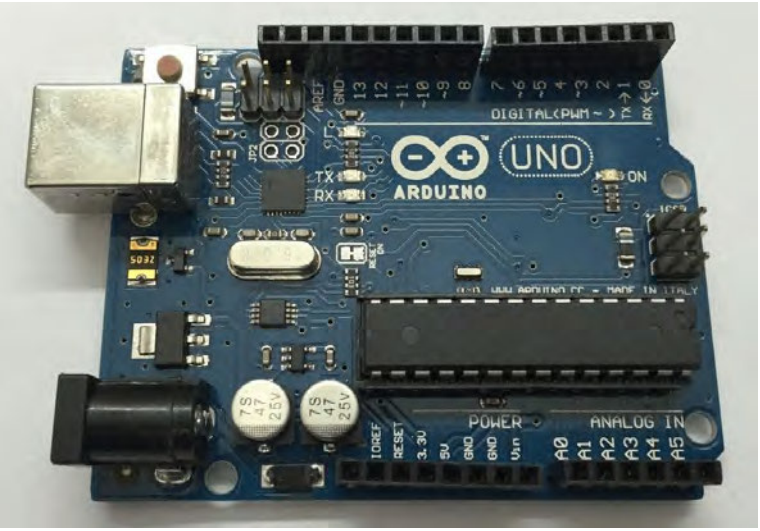


Figura 11.1: O Arduino UNO

Também precisará e dois servos motores, que podem ser pequenos, pois não será necessário carregar muito peso:

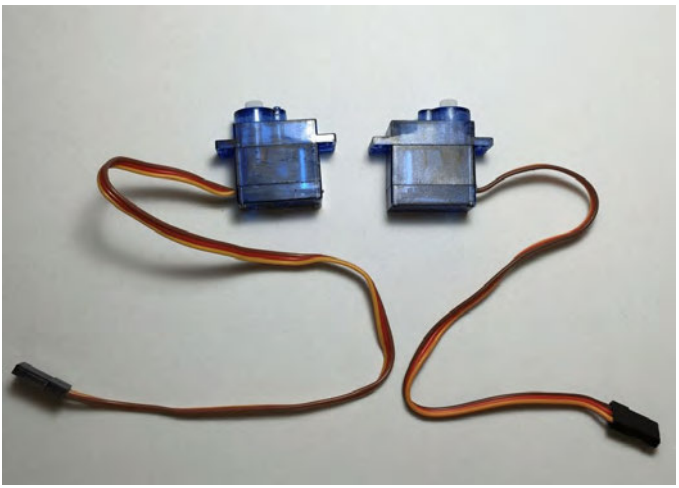


Figura 11.2: Dois servos motores

Os servos motores são fáceis de controlar usando um Arduino, basta usar a biblioteca Servo que provê todos os comandos necessários para isso. A biblioteca Servo já vem integrada ao IDE Arduino. Você pode inseri-la ao seu projeto clicando no menu `Sketch`, selecionando a opção `Incluir Biblioteca` e depois clicando sobre a opção `Servo`:

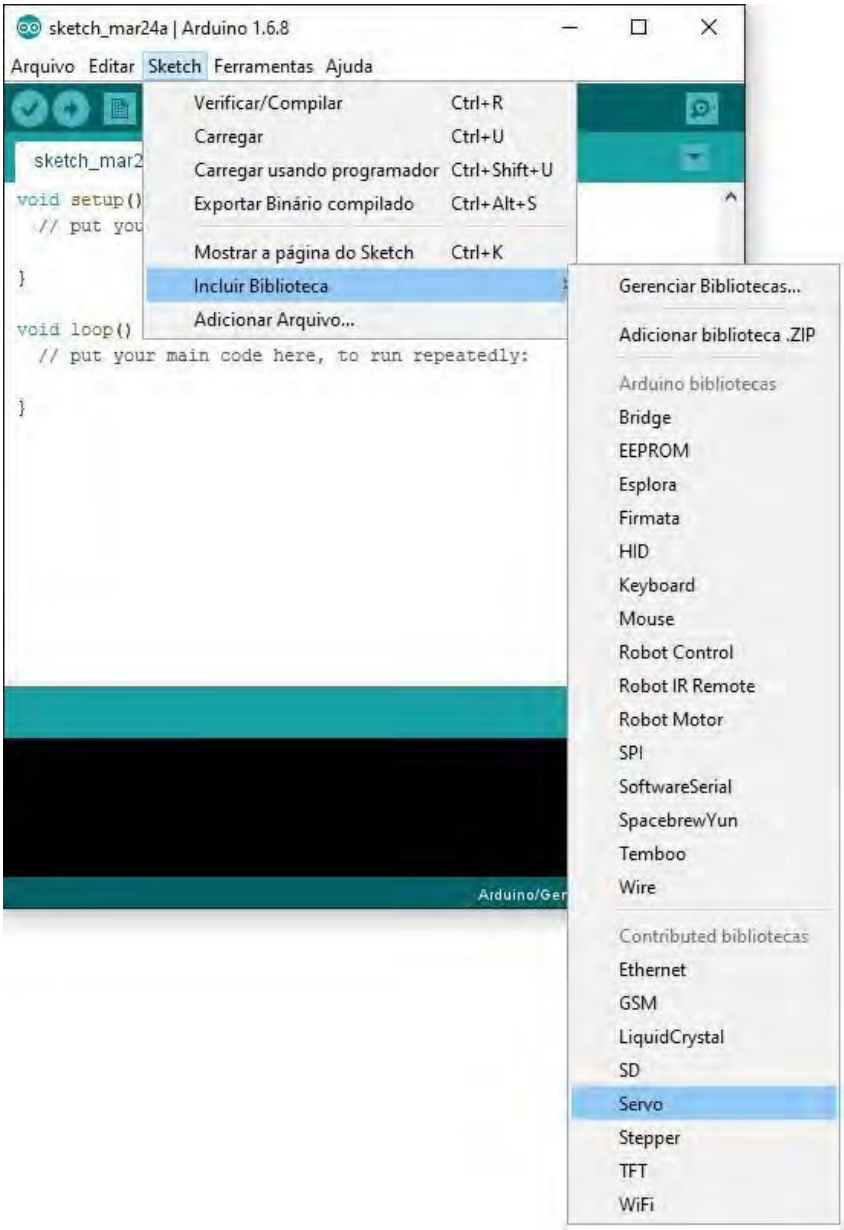


Figura 11.3: Como selecionar a biblioteca Servo para inseri-la em seu projeto

Um servo motor tem 3 fios: dois para a alimentação e um para o

sinal. Os fios de alimentação normalmente são vermelho para o +5V, e marrom para o GND. O terceiro, e normalmente amarelo, é o fio para dados, ou seja, controle do motor.

Para ligar um servo motor ao Arduino, basta conectar os fios descritos nos pinos corretos: ligue o fio vermelho a uma alimentação de +5V, o fio marrom em um GND e o fio amarelo ao pino digital 2. É importante saber que a corrente usada por um servo motor pode ultrapassar 1A, ou seja, ser muito superior ao disponível em Arduino. Portanto, **nunca** ligue mais de um servo motor a seu Arduino ao mesmo tempo. Isso pode danificar tanto o microcontrolador quanto os motores.

Vamos alimentar nossos servos através de um conjunto de quatro pilhas, que resultará em 6V, dentro do tolerado pelos servos e também útil para alimentar o próprio Arduino.

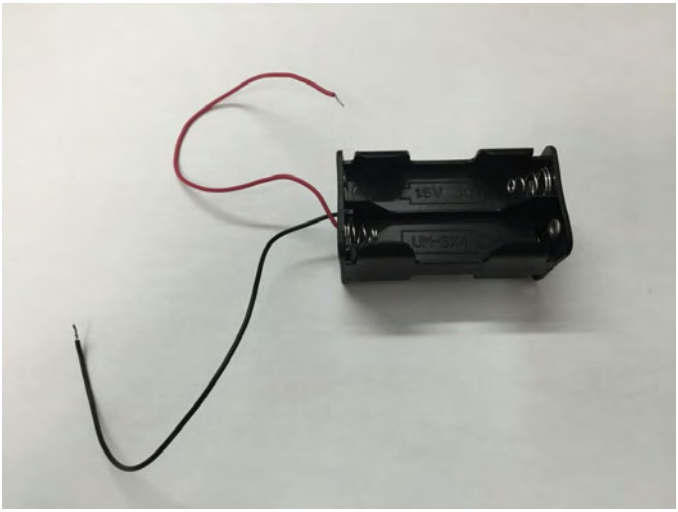


Figura 11.4: O suporte para 4 pilhas de 1,5V totalizando 6V

Para conseguir alimentar o Arduino e os motores, usarei um plug com bornes. Assim, conseguiremos derivar os fios do suporte para pilhas para os motores mantendo o Arduino ligado.

Figura 11.5: O plug com borne

Logo, é só conectar ao borne os fios do suporte de pilhas e os fios para alimentar os motores.

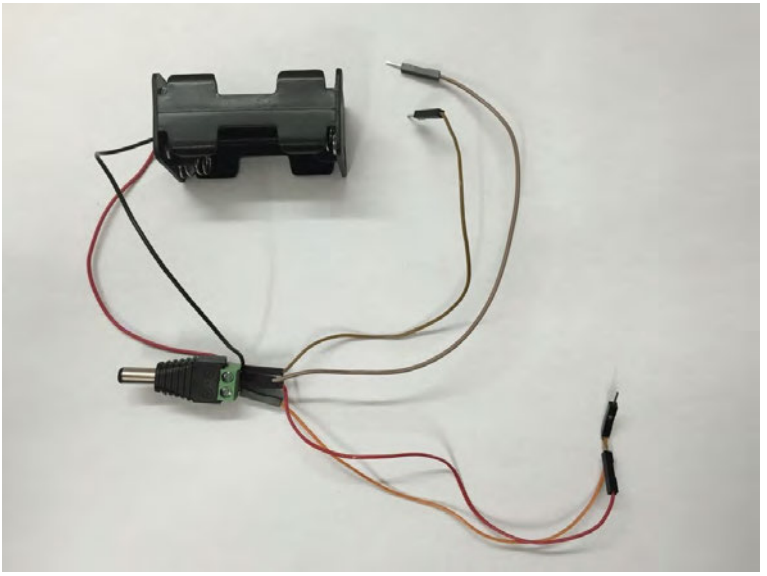


Figura 11.6: Suporte de pilhas e fios para os motores

Agora podemos ligar os fios de controle dos motores ao Arduino para testá-los. Ligue um dos motores ao pino digital 2 e o outro ao pino digital 3:

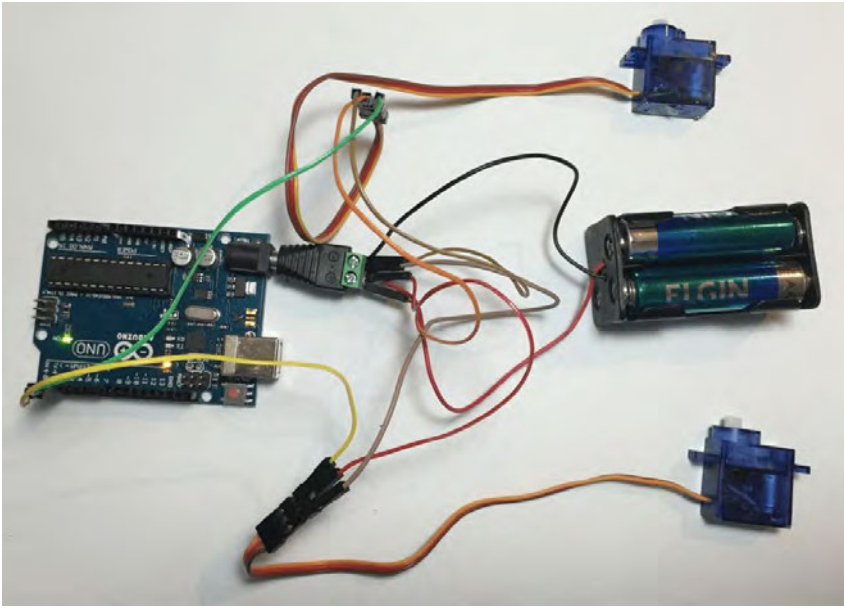


Figura 11.7: Servos motores ligados ao Arduino

Agora podemos testar o funcionamento dos motores com o seguinte código:

```
#include <Servo.h>

Servo motor_d;
Servo motor_t;

int dianteira = 2;
int traseira = 3;

int i = 0;

void setup() {
  motor_d.attach(dianteira);
  motor_t.attach(traseira);
}
```

```
void loop() {
  motor_d.write(i);
  motor_t.write(i);
  i++;
  if(i > 180) i = 0;
  delay(100);
}
```

Nesse código, importamos a biblioteca `Servo.h` que provém muitas facilidades para movermos os servos motores. Logo em seguida, declaramos dois objetos, chamados `motor_d` (motor dianteiro) e `motor_t` (motor traseiro), como sendo do tipo `Servo`.

As variáveis inteiras `dianteira`, que recebe 2, e `traseira`, que recebe 3, indicam os pinos digitais do Arduino em que estarão ligados os servos motores. Isso facilita caso queira, por um motivo qualquer, mudar os motores de pinos.

A variável inteira `i`, que recebe 0, servirá de contadora para passarmos aos motores os ângulos em que queremos posicioná-los. Na função `setup()`, usamos os comandos `motor_d.attach(dianteira)` e `motor_t.attach(traseira)` para indicar em que pino cada servo motor está ligado. Esse comando atribui o pino digital ao objeto que foi designado para o servo.

Já na função `loop()`, usamos `motor_d.write(i)` e `motor_t.write(i)` para enviar (escrever) para o servo motor o ângulo no qual queremos posicioná-los. Como a variável `i` tem o valor inicial zero, após a execução dessas linhas, os motores estarão posicionados em 0°. Isso será abrupto e deverá fazer um ruído bem característico.

Continuando, incrementamos uma variável `i` com `i++`. Em seguida, verificamos se o valor dessa variável é maior do que 180 com `if(i > 180)`. Note que o ângulo máximo de um servo motor

é 180°. Caso esse valor seja atingido, voltaremos a variável `i` para zero, seu valor inicial.

O `delay(100)` é apenas para que tudo não aconteça muito rápido e você possa perceber as mudanças graduais de ângulo. Como mencionei, o motor será posicionado em todos os ângulos entre 0° e 180° gradualmente a cada 100 milissegundos. Mas depois de atingir o valor máximo, voltará de uma vez para a posição inicial em 0°.

Já pensando no movimento das pernas, seria melhor se essa volta também fosse gradual. Vamos alterar o código anterior:

```
#include <Servo.h>

Servo motor_d;
Servo motor_t;

int dianteira = 2;
int traseira = 3;

int i = 0;
int sentido = 0;

void setup() {
  motor_d.attach(dianteira);
  motor_t.attach(traseira);
}

void loop() {
  motor_d.write(i);
  motor_t.write(i);
  if(sentido == 0) i++; else i--;
  if(i > 180) sentido = 1;
  if(i < 0) sentido = 0;
  delay(100);
}
```

Criamos apenas a variável inteira chamada `sentido` e inicializamo-la com 0. Ela indicará em qual sentido o servo motor deve ser movido. Ela define que, quando for 0, os ângulos aumentarão, e quando foi 1, os ângulos diminuirão.

Tudo é resolvido com três estruturas de seleção:

- Com `if(sentido == 0) i++; else i--;` , verificamos o sentido como acabei de mencionar: se zero, aumenta os ângulos; se não, diminui.
- Com `if(i > 180) sentido = 1;` , verificamos se o ângulo máximo foi atingido e, caso tenha, indicamos a inversão do sentido, mudando a variável `sentido` .
- Com `if(i < 0) sentido = 0;` , verificamos se o ângulo mínimo foi atingido, caso tenha, indicamos a inversão do sentido, mudando a variável `sentido` .

Testando você verá que o eixo do servo motor vai de um lado até outro gradualmente, sem aquela mudança brusca que acontecia antes.

Vamos mudar o código anterior novamente para o que segue. Esse próximo código coloca os servos motores na posição de 90º, ou seja, no meio do percurso. Isso será importante para colocar as pernas na posição inicial.

```
#include <Servo.h>

Servo motor_d;
Servo motor_t;

int dianteira = 2;
int traseira = 3;

void setup() {
    motor_d.attach(dianteira);
    motor_t.attach(traseira);
    motor_d.write(90);
    motor_t.write(90);
}

void loop() {
}
```

O funcionamento é idêntico ao código anterior, porém criamos

os objetos para os motores, e determinamos variáveis para representar em quais pinos digitais estão os motores. Já na função `setup()`, anexamos os motores aos seus pinos, e usamos `motor_d.write(90)` e `motor_t.write(90)` para posicionar os motores em 90°.

Feito tudo isso é hora de construir as pernas! Você vai precisar de dois clips nº 10.

Usar clips torna tudo mais simples, pois não precisaremos correr atrás de pedaço de arame ou ferro. Também não precisa ser necessariamente o clips nº 10. Você poderá fazer com clips menores, que fará com que as pernas fiquem mais curtas, ou com clips maiores, que fará com que as pernas fiquem mais longas. Faça a adaptação que achar necessária.

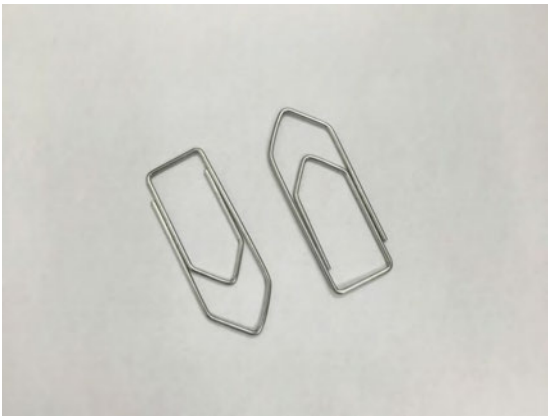


Figura 11.8: Dois clips nº 10

Abra os clips para que fiquem o mais reto possível.

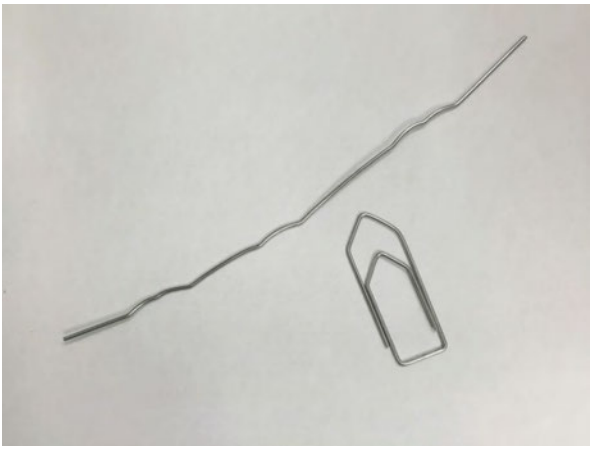


Figura 11.9: O clips aberto

Pegue o cabeçote redondo do seu servo motor e coloque o clips sobre ele, de forma que fique no meio do clips.

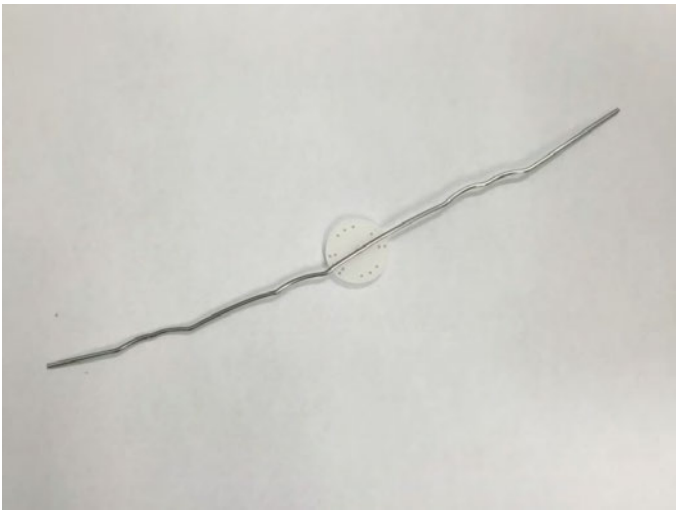


Figura 11.10: O cabeçote no meio do clips

Dobre o clips e faça uma ponta, como na figura a seguir:

Figura 11.11: Clips dobrado sobre o cabeçote

Agora você deve prender o clips no cabeçote, o que criará as nossas pernas. Você pode usar fita adesiva, cola ou, como eu, um pequeno e fino fio de cobre para amarrar tudo.

Figura 11.12: Clips, cabeçote e fio para amarrar tudo

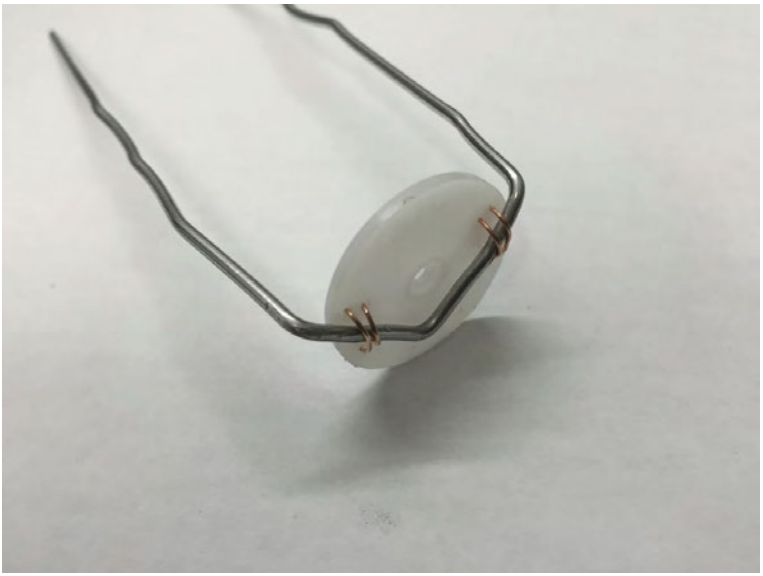


Figura 11.13: A primeira perna montada

Depois repita novamente para criar uma segunda perna.

Figura 11.14: As duas pernas montadas

Essas serão nossas pernas dianteiras, que serão colocadas no que será o motor dianteiro e traseira, colocada no que será o motor traseiro.

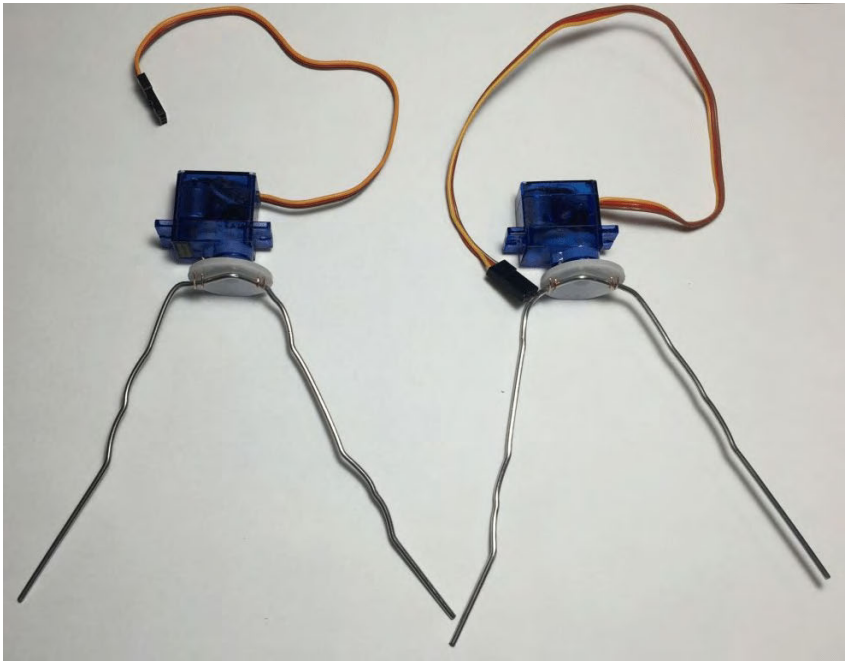


Figura 11.15: Pernas instaladas nos motores

Então vamos montar o corpo! Usando fita dupla face ou cola, prenderemos os motores em baixo do Arduino:

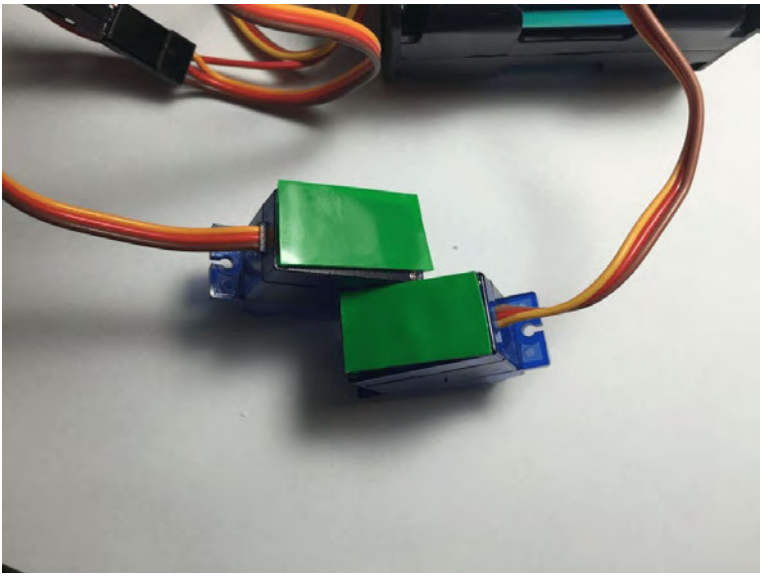


Figura 11.16: Motores com fitas dupla face

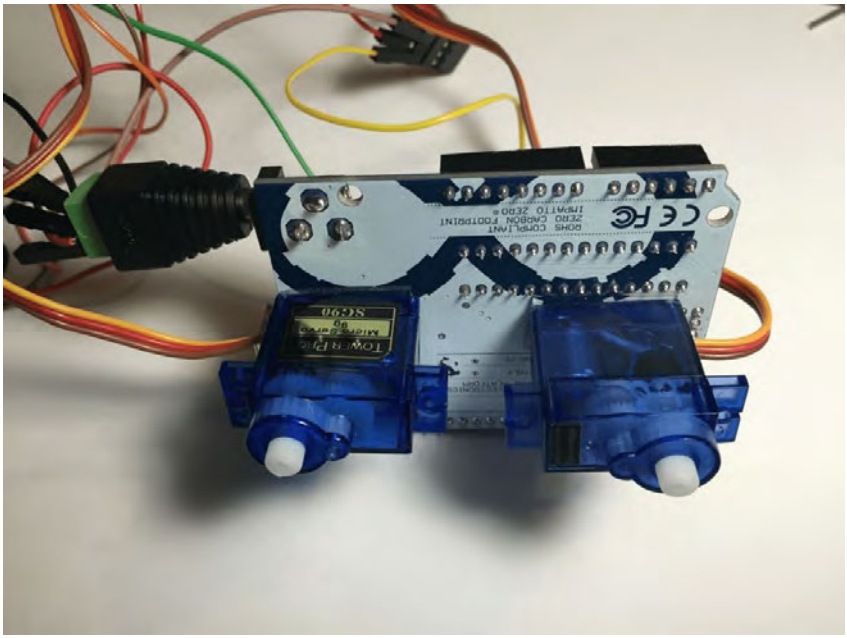


Figura 11.17: Motores colados em baixo do Arduino

Hora de colocar as pernas nos motores:

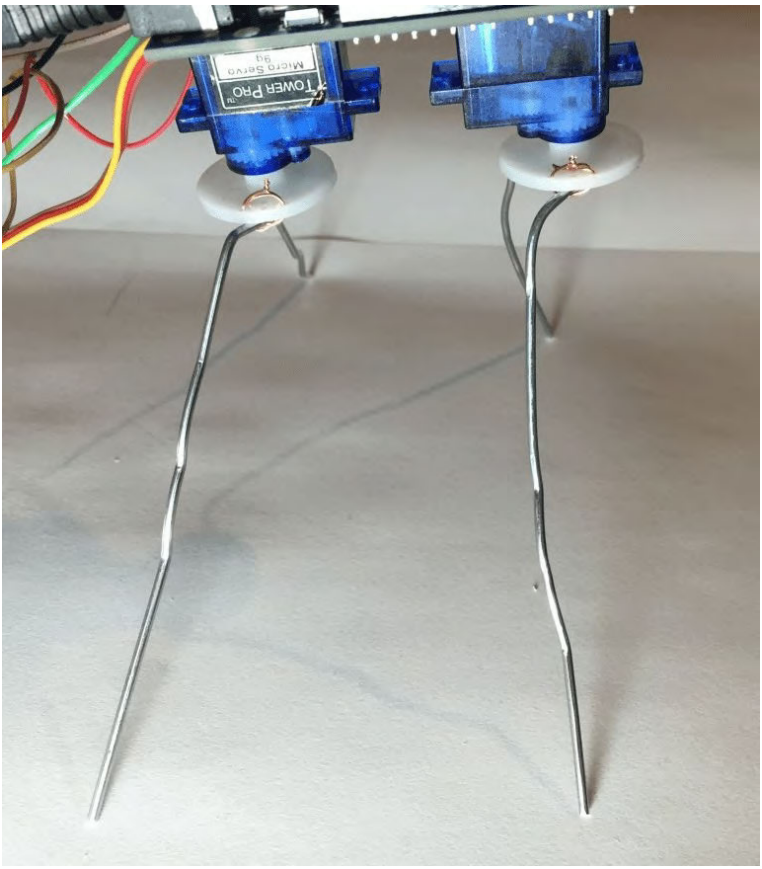


Figura 11.18: Pernas instaladas

Agora, ajeitando as pernas para que fiquem equilibradas, coloque seu robô aranha em pé. Lembre-se de aparafusar o cabeçote ao motor para que tudo se mantenha montado quando estiver se movimentando.

O código para que o robô comece a caminhar é o que segue:

```
#include <Servo.h>

Servo motor_d;
Servo motor_t;
```

```
int dianteira = 2;
int traseira = 3;

int i = 45, j = 90;
int sentido = 0;

void setup() {
  motor_d.attach(dianteira);
  motor_t.attach(traseira);
}

void loop() {
  motor_d.write(i);
  motor_t.write(j);
  if(sentido == 0) {
    i++;
    j--;
  } else {
    i--;
    j++;
  }
  if(i > 90) sentido = 1;
  if(i < 45) sentido = 0;
  delay(50);
}
```

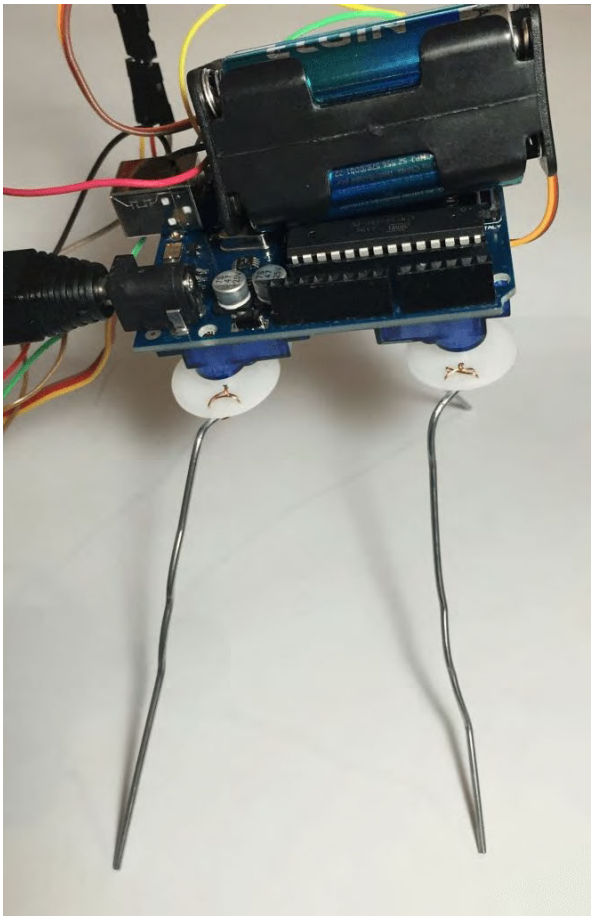


Figura 11.19: Tudo em pé

Você encontrará um vídeo desse projeto em funcionamento em https://youtu.be/q8gRwHqC6No.

Ele é um pouco desengonçado, né?! Mas vale como um bom estudo de movimentos de pernas para um projeto mais elaborado e completo.

11.3 DESAFIO

Bons desafios para esse projeto são incrementá-lo para deixá-lo mais atraente e prático. Para começar, acho que é preciso deixá-lo menos desengonçado e com movimentos mais ágeis e rápidos.

Você também pode criar seu próprio Arduino, para deixar tudo pequeno para que possa ser colocado junto aos servos motores dentro de uma pequena caixa. Lembre-se da bateria, que é praticamente o mais pesado de todo o projeto, que deve ficar também em algum lugar. As pernas talvez precisem ser reformuladas para aguentar o peso dela.

Finalmente, acho que incluir dois LEDs para criar olhos também deixará o nosso pequeno inseto mais divertido, ainda mais em locais escuros. Se bem que esses LEDs podem acender apenas quando o ambiente está escuro. Isso será fácil de se fazer usando um sensor de luminosidade.

Com esse projeto, chegamos ao fim deste livro. Mas não ao fim das possibilidades que o Arduino, quando unido a componentes eletrônicos, sensores e uma boa dose de imaginação, nos proporcionam. Siga em frente!

Use esses projetos como pontapé inicial para outros. Crie os seus a partir de apenas uma ideia, mas não pare, mantenha-se em movimento sempre.

Capítulo 12

LISTA GERAL DE MATERIAIS

A seguir, uma lista de materiais completa separada por projetos, para que seja possível você se organizar para obter todos os materiais (ou parte deles) em quantidade. Isso pode baratear o custo dos projetos.

Projeto nº 01 — Criando nosso próprio Arduino

- 1 x Arduino UNO R3
- 1 x Prot-o-board
- 1 x Suporte para baterias de 9V
- 1 x Bateria de 9V
- 1 x Cristal oscilador de 16MHz
- 1 x Resistor de 10 KΩ 1/4W
- 1 x Resistor de 330 Ω 1/4W
- 2 x Capacitores cerâmicos de 22 pF/50
- 1 x Regulador de tensão 7805
- 1 x LED vermelho

- Fios diversos

Projeto nº 02 — Automatizando uma porta com senha via teclado

- 1 x Arduino UNO
- 1 x Teclado matricial de membrana
- 1 x Transistor TIP 102
- 1 x Resistor de 1KΩ
- 1 x Relê 5V
- 1 x Fechadura elétrica com fonte 12V
- Fios diversos

Projeto nº 03 — Criando um radar para verificar a velocidade de um objeto

- 1 x Arduino UNO
- 1 x Sensor ultrassônico
- 1 x Botão
- 1 x Resistor 1KΩ 1/4W
- 1 x Display de 7-segmentos com dois dígitos
- 2 x Resistores 330Ω 1/4W
- 1 x Suporte para baterias 9V com plug para o Arduino
- 1 x Bateria de 9V
- Fios diversos.

Projeto nº 04 — Que tal acionar as lâmpadas de sua casa com controle remoto?

- 1 x Arduino UNO R3
- 1 x Prot-o-board
- 1 x Sensor infravermelho
- 1 x Controle remoto
- 1 x Resistor de 1 KΩ 1/4W
- 1 x Transistor TIP102
- 1 x Relê 5V/125V
- 1 x Abajur (110V com lâmpada de 7W)
- 1 x Ferro de solda
- Estanho para solda
- Fios diversos

Projeto nº 05 — Um dado eletrônico para jogos

- 1 x Arduino
- 9 x LEDs vermelhos
- 3 x Resistores de 220Ω
- 1 x Ferro de solda
- Estanho para solda
- Fios diversos

Projeto nº 06 — Criando um videogame para você

- 1 x Arduino UNO
- 1 x Prot-o-board
- 1 x Resistor de 470Ω
- 1 x Resistor de 1KΩ
- 1 x Potenciômetro de 10KΩ
- 1 x Conector RCA macho
- 1 x Ferro de solda
- Estanho para solda
- Cabo com pelo menos duas vias (fios)
- Fios diversos

Projeto nº 07 — Alarme de geladeira com monitoramento de abertura da porta

- 1 x Arduino UNO
- 1 x Prot-o-board
- 1 x Sensor magnético para alarmes (Reed-switch)
- 1 x Resistor 1KΩ 1/4W
- 1 x Buzzer
- 1 x Fonte para telefone celular 5V 1A USB
- 1 x Cabo USB
- Fios diversos

Projeto nº 08 — Construindo um rastreador GPS

offline

- 1 x Arduino UNO
- 1 x Módulo GPS
- 1 x Módulo cartão SD
- 1 x Cartão SD
- Fios diversos

Projeto nº 09 — Batata quente, quente... quente... queimou!

- 1 x Arduino UNO R3
- 1 x Buzzer
- 1 x Suporte para baterias 9V com plug para o Arduino
- 1 x Bateria 9V

Projeto nº 10 — O robô aranha mais simples do mundo!

- 1 x Arduino UNO
- 2 x Servos motores
- 3 x Clips nº 10
- 1 x Suporte para 4 pilhas
- 4 x Pilhas
- Fios diversos