Competências, Habilidades e Bases Tecnológicas da disciplina de Sistemas Embarcados.

II.4 SISTEMAS EMBARCADOS

Função: Desenvolvimento de aplicações para sistemas embarcados
Classificação: Execução

Atribuições e Responsabilidades

Desenvolver sistemas embarcados.

Valores e Atitudes

Incentivar a criatividade.
Estimular a organização.
Estimular o interesse na resolução de situações-problema.

Competências	Habilidades
1. Analisar modelos de sistemas embarcados.	1.1 Identificar as características de sistemas embarcados.
2. Desenvolver aplicações com microcontroladores.	2.1 Programar sistemas para microcontroladores. 2.2 Executar instruções para microcontroladores.

Bases Tecnológicas

Introdução desenvolvimento de *software* embarcado
- *Hardware open-source;*
- Movimento *maker* e *tinkering;*
- Internet das coisas;
- Microcontroladores de Sistemas embarcados;
- Linguagem, IDE e simuladores.

Princípios de elétrica e eletrônica
- Práticas de manuseio de componentes;
- Protoboards, LEDs e botões;
- Módulos e *shields*.

Programação de microcontroladores
- Estrutura de um programa (*setup* e *loop*);
- Compilação, gravação e execução;
- Variáveis e tipos de dados;
- Estruturas de decisão e repetição;
- Funções.

Entrada e saída digital
- *pinMode*, *digitalWrite* e *digitalRead*.

Utilização de controle de tempo
- *Timers* e contadores;
- Millis e micros;
- *Delay* e *delayMicroseconds*.

Entrada e saída analógica
- Conversão Analógico-Digital e Digital-Analógico.
- *AnalogReference*, *analogRead* e *analogWrite*.

Funções
- Funções matemáticas e trigonométricas e de texto;
- Números aleatórios.

Bibliotecas
- Sensores, sons, interrupções e comunicação.

Carga horária (horas-aula)

Teoria	00	**Prática Profissional**	60	**Total**	**60 Horas-aula**
Teoria (2,5)	00	**Prática Profissional (2,5)**	50	**Total (2,5)**	**50 Horas-aula**

Possibilidade de divisão de classes em turmas, conforme o item 4.8 do Plano de Curso.

Todos os componentes curriculares preveem prática, expressa nas habilidades relacionadas às competências. Para este componente curricular, está prevista divisão de classes em turmas.

Para ter acesso às titulações dos profissionais habilitados a ministrarem aulas neste componente curricular, consultar o site: https://crt.cps.sp.gov.br/index.php

Observações a considerar:

1-) Comunicação de alunos com alunos e professores:

- Microsoft Teams;
- Criação de grupo nas redes sociais também é interessante.

2-) Uso de celulares:

Para o bom andamento das aulas, recomendo que utilizem os celulares em *vibracall*, para não atrapalhar o andamento da aula.

3-) Material das aulas:

A disciplina trabalha com **NOTAS DE AULA** que são disponibilizadas ao final de cada aula.

4-) Prazos de trabalhos e atividades:

Toda atividade solicitada terá uma data limite de entrega, de forma alguma tal data será postergada ou seja, se não for entregue até a data limite a mesma receberá menção I, isso tanto para atividades entregues de forma impressa, digital ou no Microsoft Teams.

5-) Qualidade do material de atividades:

- Impressas ou manuscritas:
  Muita atenção na qualidade do que será entregue, atividades sem grampear, faltando nome e número de componentes, rasgadas, amassadas, com rebarba de folha de caderno e etc. serão desconsiderados por mim.

- Digitais:
  Ao enviarem atividades para o e-mail, ou inseridas no Microsoft Teams, **SEMPRE** deverá ter o nome da atividade que está sendo enviada, e no corpo do e-mail ou da atividade deverá ter o(s) nome(s) do(s) integrante(s), sem estar desta forma a atividade será **DESCONSIDERADA**.

6-) Menções e critérios de avaliação:

Na ETEC os senhores serão avaliados por MENÇÃO, onde temos:
- MB - Muito Bom;
- B - Bom;
- R - Regular;
- I - Insatisfatório.

Cada trimestres poderemos ter as seguintes formas de avaliação:
- Avaliação teórica;
- Avaliação prática (a partir do 2º trimestre, e as turmas do 2º módulo em diante);
- Seminários;
- Trabalhos teóricos e/ou práticos;
- Assiduidade;
- Outras que se fizerem necessário.

Caso o aluno tenha alguma menção I em algum dos trimestres será aplicada uma recuperação, que poderá ser em forma de trabalho prático ou teórico.

1. Recomendações bibliográficas:

KARVINEN, Kimmo; KARVINEN, Tero. Primeiros Passos com Sensores. Primeira edição. São Paulo: Novatec, Outubro 2014.

BANZI, Massimo; SHILOH, Michael. Primeiros Passos com o Arduino. Segunda edição. São Paulo: Novatec, Maio 2015.

2. *O que é Open Source Hardware?*

De modo geral, os Open Hardwares são plataformas cujo design, código ou tecnologia podem ser replicados, customizados e distribuídos gratuitamente. Ele é parte fundamental da cultura *maker*, que tem a colaboração e a sustentabilidade como dois de seus pilares fundamentais.

Existem dois tipos de licenças, com características específicas, as *copyleft* e as permissivas. Vamos falar mais sobre elas abaixo, mas, de modo geral, elas seguem os seguintes princípios: Hardware pode ser modificado, usado comercialmente e distribuído; hardware pode ser modificado e usado de forma privada; a licença e os direitos precisam ser incluídos no hardware; os autores não provêm garantias.

2.1. *Hardware* ou *software*?

Os *softwares* livres ficaram muito populares no início nos anos 2000. O Linux, por exemplo, é um sistema operacional de software livre, que pode ser distribuído gratuitamente. Embora a ideia do Open Hardware seja similar à do software livre, são coisas diferentes. O hardware é a parte física da máquina, as peças que a compõem, como as placas operacionais e de memória. O software, diferentemente, engloba os programas que fazem com que o computador, como aplicativos e sistemas operacionais.

3. Introdução ao Arduino

O Arduino faz parte do conceito de hardware e software livre e está aberto para uso e contribuição de toda sociedade. O conceito Arduino surgiu na Itália, em 2005, com o objetivo de criar um dispositivo que fosse utilizado em projetos/protótipos construídos de uma forma menos dispendiosa do que outros sistemas disponíveis no mercado.

Ele pode ser usado para desenvolver artefatos interativos *stand-alone* ou conectados ao computador, utilizando diversos aplicativos, tais como: *Adobe Flash*, *Processing, Max/MSP, Pure Data ou SuperCollider.*

O Arduino foi projetado com a finalidade de ser de fácil entendimento, de fácil programação e de fácil aplicação, além de ser multiplataforma, podendo ser configurado em ambientes Linux, *Mac OS* e *Windows*. Além disso, um grande diferencial deste dispositivo é ser mantido por uma comunidade que trabalha na filosofia *open-source*, desenvolvendo e divulgando gratuitamente seus projetos.

O equipamento é uma plataforma de computação física: são sistemas digitais ligados a sensores e atuadores, que permitem construir sistemas que percebam a realidade e respondem com ações físicas. Ele é baseado em uma placa microcontrolada, com acessos de Entrada/Saída (I/O), sobre a qual foram desenvolvidas bibliotecas com funções que simplificam a sua programação, por meio de uma sintaxe similar a das linguagens C e C++.

O Arduino utiliza o microcontrolador Atmega. Um microcontrolador (também denominado MCU) é um computador em um chip, que contém um microprocessador, memória e periféricos de entrada/saída. Ele pode ser embarcado no interior de algum outro dispositivo, que, neste caso, é o Arduino, para que possa controlar suas funções ou ações.

Em resumo, o Arduino é um kit de desenvolvimento, que pode ser visto como uma unidade de processamento capaz de mensurar variáveis do ambiente externo, transformadas em um sinal elétrico correspondente, através de sensores ligados aos seus terminais de entrada. De posse da informação, ele pode processá-la computacionalmente. Por fim, ele pode ainda atuar no controle ou no acionamento de algum outro elemento eletroeletrônico conectado ao terminal de saída. A Figura abaixo apresenta um diagrama de blocos de uma cadeia de processamento utilizando o Arduino.

Uma vez que o Arduino é baseado em um microcontrolador e, portanto, é programável, torna-se possível criar diversas aplicações diferentes com uma certa facilidade. Além disso, o próprio equipamento pode ser reutilizado, através de uma nova programação. Por sua vez, a sua programação é simplificada pela existência de diversas funções que controlam o dispositivo, com uma sintaxe similar a de linguagens de programação comumente utilizadas (C e C++).

Inicialmente vou mostrar como instalar o mesmo no computador para que se possa fazer a interação física do Arduino, e mostrar como funciona na IDE Arduino do computador, após isso, irei mostrar no simulador TinkerCad.

4. Anatomia de uma placa Arduino

4.1. Portas Digitais

O Arduino Uno oferece 14 portas digitais que podem ser utilizadas tanto para entrada (input) como para saída (output) e que podem ser utilizadas para comandar 14 dispositivos externos. Estas portas vão de 0 a 13, observe com atenção a figura:

4.2. Portas PWM

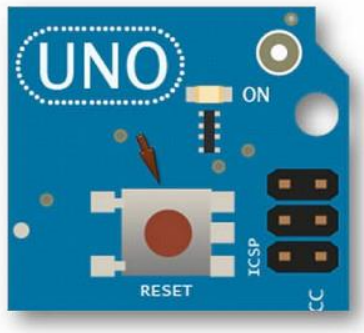
Estas portas simulam uma porta analógica e são em número de seis, note que elas além do número da porta têm também uma pequena onda senoidal como mostrado na figura. As portas digitais que também podem ser usadas como portas PWM são as de número: 11, 10, 9, 6, 5 e 3. Mas tenha em mente que elas também são portas digitais.

4.3. Portas RX e TX

Essas duas portas embora possam ser utilizadas como Portas digitais também são utilizadas pelo Arduino para comunicação serial tanto para entrada como para saída de dados. É conveniente evitar uso destas portas como portas digitais, mas não proibido.

4.4. Reset

Este botão, mostrado na figura abaixo, tem como única função reinicializar o Arduino Uno, mas, note que ele muda um pouco seu posicionamento em outras placas.



4.5. Microcontrolador

É onde tudo acontece, é o cérebro desta do Arduino, onde fica gravado o código desenvolvido e que será executado, mas note que quando se grava um código o anterior é descartado, sempre fica apenas o último código gravado.

Este processador permite que o Arduino funcione de forma autônoma, ou seja, uma vez transferido o código para ele não existe mais a necessidade de uma conexão com o computador. A figura 1 mostra o tipo de processador mais comumente encontrado, mas existem outros como mostrado na figura 2. A grande vantagem do processador da figura 1 é a facilidade com que pode ser removido e transferido para outra placa.

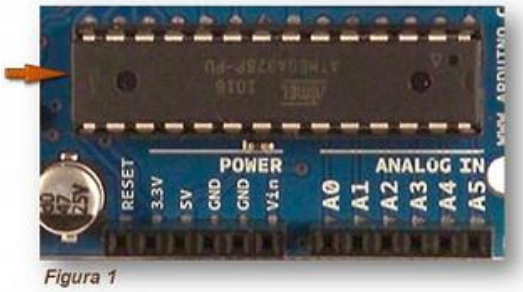


Figura 1

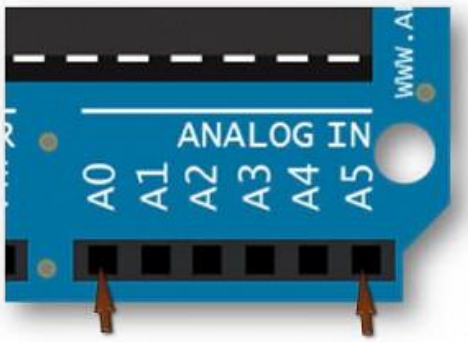
Figura 2

4.6. Portas Analógicas

Essas portas são unicamente para entrada de dados e comumente usadas para comunicação com sensores que podem ser utilizados para determinar:

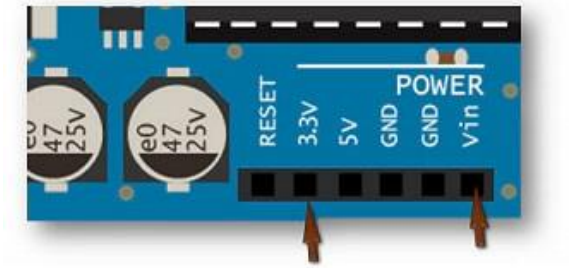
- Temperatura;
- Quantidade de luz;
- Umidade;
- Dentre outras ações.

As portas analógicas são em número de 6 e vão de A0 até A5.



4.7 Pinos de Energia

Estes pinos fornecem energia para dispositivos externos como mostrado na figura ao lado e explicados abaixo: Observe com atenção a figura, ela mostra estes pinos.

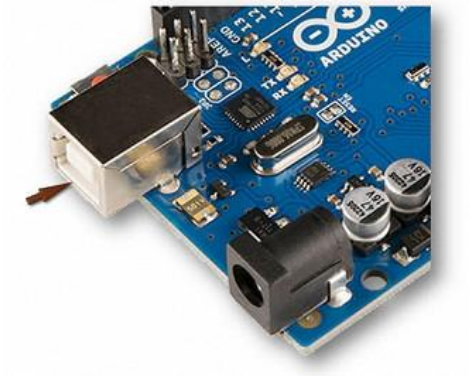


- 3,3V: este pino fornece a 3.3 volts a dispositivos externos e está marcado na placa;
- 5V: fornece 5 volts a dispositivos externos e também está indicado na placa.
- GND: fornece potencial de terra a dispositivos externos, o seja 0 volt e são em número de dois e como pode observar também se encontram bem identificado na placa.
- Vin: este pino fornece ao dispositivo externo a mesma tensão que está sendo recebida pelo Pino de alimentação externa. Note que é o último pino mostrado na figura.

Deve ser observado que estes pinos fornecem uma corrente muito baixa, portando devem ser usados para pequenas cargas e nunca para ligar um motor, por pequeno que ele possa ser dentre outros dispositivos de alto consumo. De maneira geral aguentam alimentar apenas um LED, a base de um transistor ou coisa do gênero.

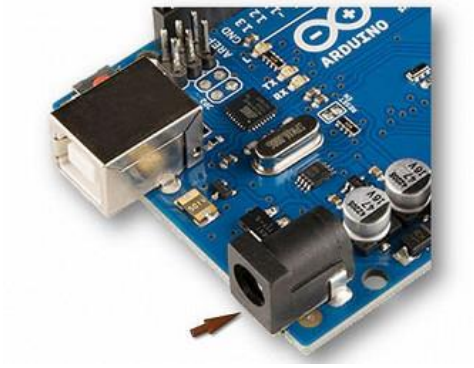
4.8. Porta USB

Esta é a porta usada para estabelecer uma conexão entre a placa de Arduino e o PC. É ela que permite o envio de códigos para o processador, permite conexão com a serial e também é usada para a alimentação da placa. Observe a figura:



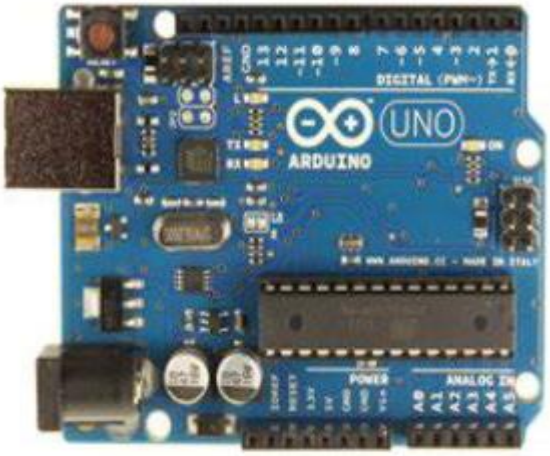
4.9. Pino para Alimentação externa

Este é o pino para a alimentação externa da placa, ou seja, quando não estiver sendo usado a porta USB para conexão com o computador, e é usando-o que vamos alimentar a placa. Ele é que permite a autonomia desta placa. Podemos alimentar esta placa com tensão entre 6 e 12 volts sem problemas.



5. Arduino em Windows

5.1 Placa Arduino e um cabo USB AB



5.2. - Download do software do Arduino

Faça download da última versão do software do Arduino(http://multilogica-shop.com/Download). Ao terminar, descompacte o arquivo e mantenha a estrutura de pastas e sub-pastas. Se quiser guarde esta pasta no drive C: do seu computador. Dentro desta pasta existe um arquivo chamado arduino.exe que é o ponto de entrada do programa do Arduino, a IDE (Integrated Development Environment).

5.3. Conectando o Arduino

O Arduino Uno isolado usa a energia do computador através da conexão USB, não sendo necessária energia externa. Conecte a placa Arduino ao computador usando o cabo USB AB. O LED verde de energia (PWR) deve acender.

5.4. Instalando os drivers

Drivers para Arduino Uno ou Arduino Mega 2560 com Windows 7, Vista ou XP:

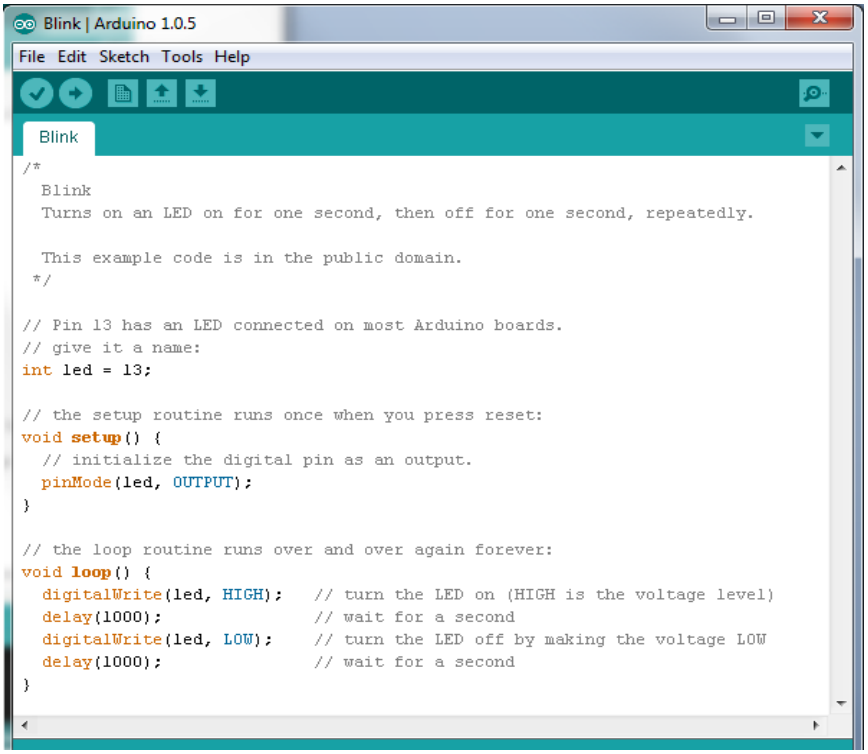
- Conecte a placa ao computador e aguarde o Windows iniciar o processo de instalação do driver. Depois de alguns momentos o processo vai falhar. Clique em concluir e dispense a ajuda do assistente.
- Clique no Menu Principal e abra o Painel de Controle.
- Dentro do Painel de Controle, navegue até Sistema e Segurança. Na sequência clique em Sistema, selecione Hardware e depois clique em Gerenciador de Dispositivos.
- Procure por Portas (COM & LPT), onde você deve ver uma opção Arduino UNO (COMxx).
- Clique com o botão da direita em Arduino UNO (COMxx) e escolha a opção Atualizar Driver.
- Depois escolha a opção Instalar de uma lista ou local específico (Avançado), e clique em avançar.
- Finalmente navegue e escolha o driver arduino.inf localizado na pasta Drivers do software do Arduino que você baixou.
- O Windows vai finalizar a instalação do driver a partir deste ponto.

5.5. Abrindo o programa Arduino

Clique duas vezes na aplicação do Arduino, o arquivo arduino.exe. Caso o programa carregue com o idioma que não é da sua preferência você pode alterar na sessão de preferências do programa.

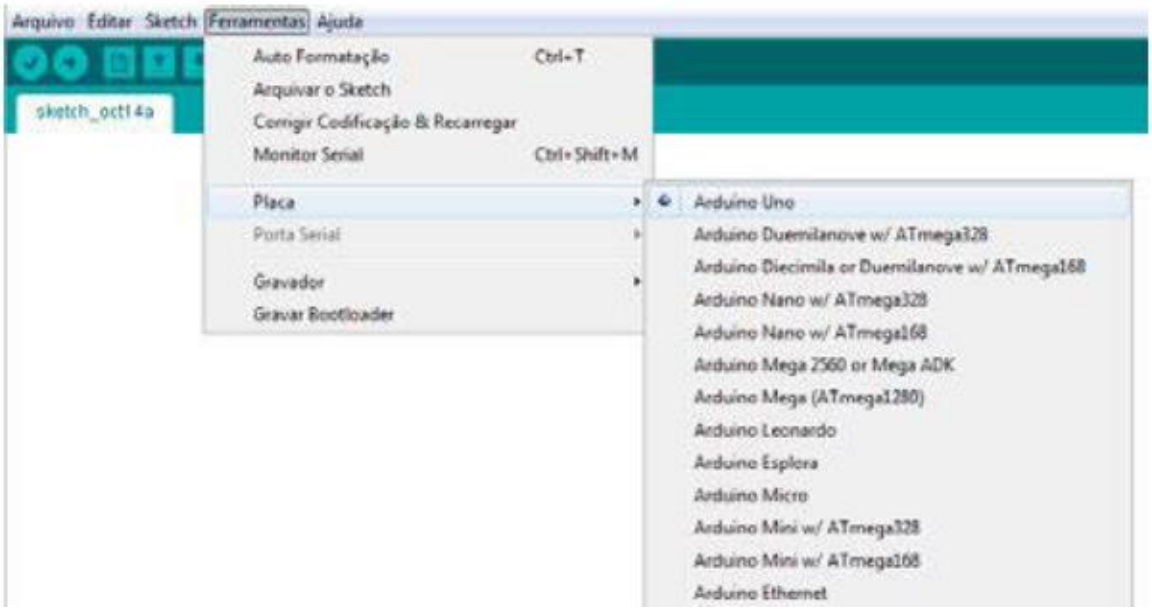
5.6. Exemplo Piscar

Abra o exemplo Piscar (blink): Arquivo > Exemplos > 01.Basics > Blink



5.7. Selecione sua placa

Você deve selecionar qual a sua placa Arduino: Ferramentas > Placa > Arduino Uno.



5.8. Selecione a porta

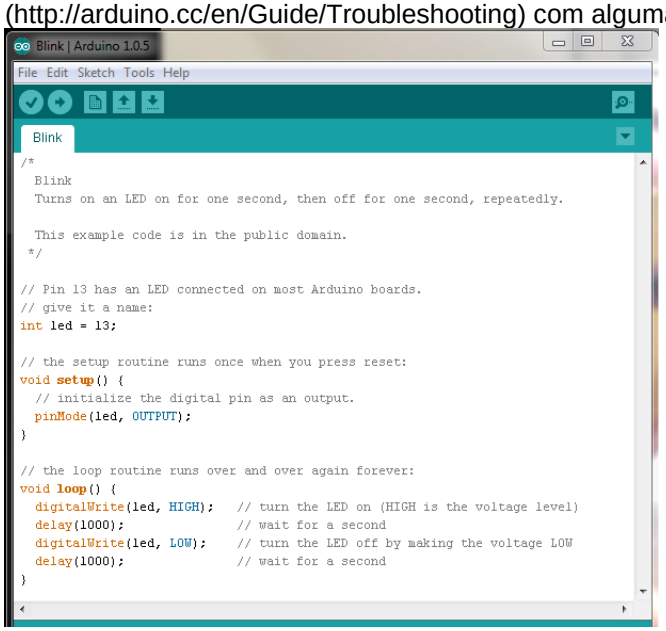
Selecione agora a porta serial que conectará o Arduino: Ferramentas > Porta Serial. Você deve selecionar a mesma porta que utilizou para configurar o sistema no passo 4.

5.9. Carregue o programa

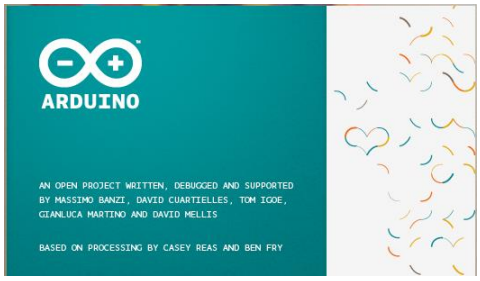
Agora simplesmente clique no botão Carregar da janela do programa. Espere alguns segundos. Você deve ver os LEDs RX e TX da placa piscarem. Se o processo foi executado normalmente você verá uma mensagem de “Transferência concluída”.

Depois de alguns segundos você verá o LED do pin 13 piscar em laranja. Neste caso, parabéns! Seu Arduino está pronto e instalado.

Se você tiver problemas na instalação pode acessar a página oficial do Arduino (http://arduino.cc/en/Guide/Troubleshooting) com algumas soluções.



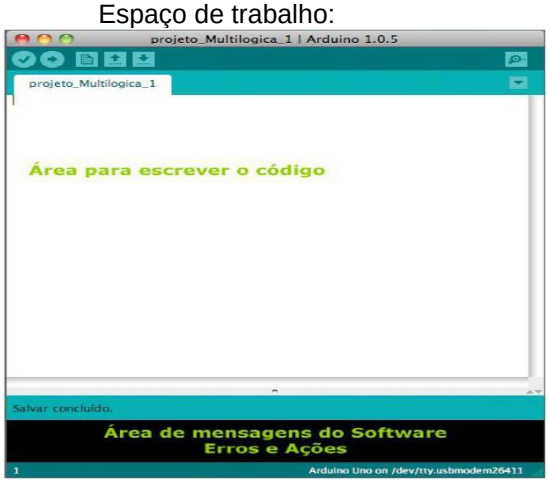
6. Software Arduino



Para executar o programa entramos na pasta do Arduino guardada no computador e procuramos o ícone. Clique duas vezes para abrir o programa.

O programa do Arduino também é conhecido como IDE Arduino (Integrated Development Environment) pois além do entorno de programação consiste também em um editor de código, um compilador e um depurador.

Espaço de trabalho:

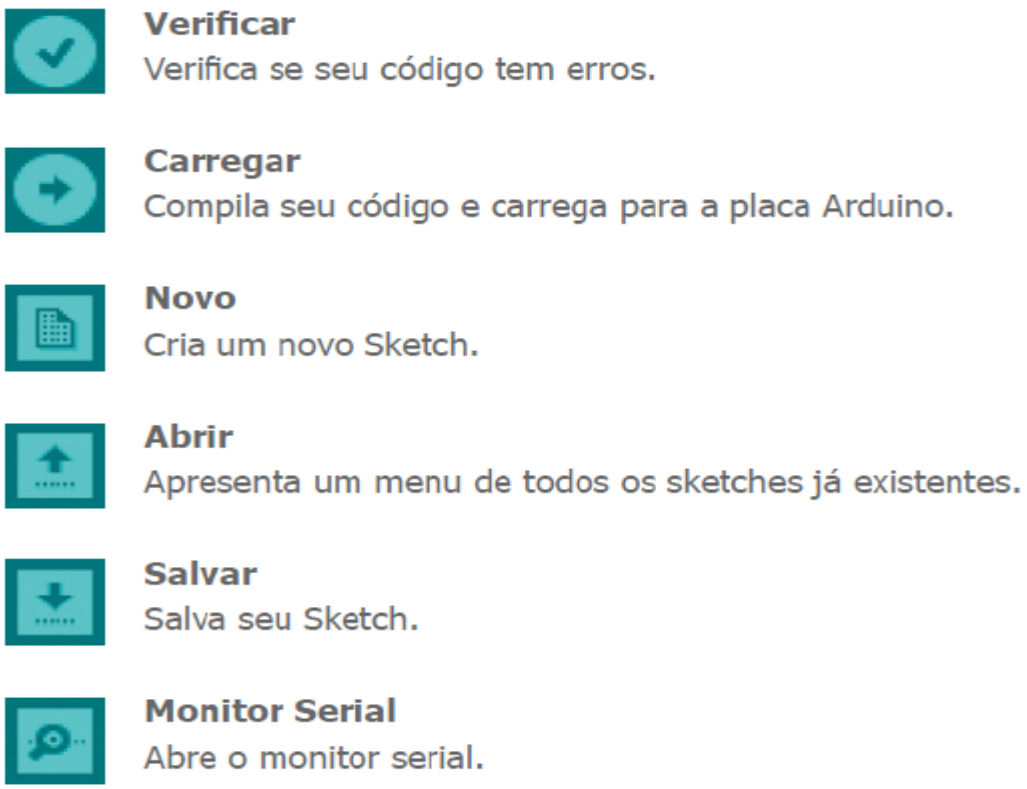


6.1. Sketches

Softwares escritos usando Arduino são chamados de *Sketches*. Estes Sketches são escritos no editor de texto da IDE do Arduino e são salvos com a extensão de arquivo .ino. Este editor tem características de cortar/colar e para buscar/substituir texto. A área de mensagem dá feedback ao salvar e exportar arquivos e também exibe informações de erros ao compilar

Sketches. O canto direito inferior da janela exibe a placa atual e a porta serial. Os botões da barra de ferramentas permitem que você verifique, carregue, crie, abra e salve Sketches ou abra o monitor serial.

Nota: Nas versões do IDE antes de 1.0 os Sketches são salvos com a extensão .pde. É possível abrir esses arquivos com a versão 1.0, mas você será solicitado a salvar o Sketch com a extensão .ino.



6.2. Biblioteca Arduino

O ambiente Arduino pode ser estendido através da utilização de bibliotecas, assim como a maioria das plataformas de programação. Bibliotecas fornecem funcionalidades extras para uso em sketches. Por exemplo, para trabalhar com hardware ou manipulação de dados.

Algumas bibliotecas já vêm instaladas com a IDE Arduino, mas você também pode fazer download ou criar a sua própria.

Para usar uma biblioteca em um sketch, selecione em sua IDE Arduino:

Sketch> Importar Biblioteca.

Dentro da programação você inclui as funcionalidades de uma biblioteca já existente a partir do comando:

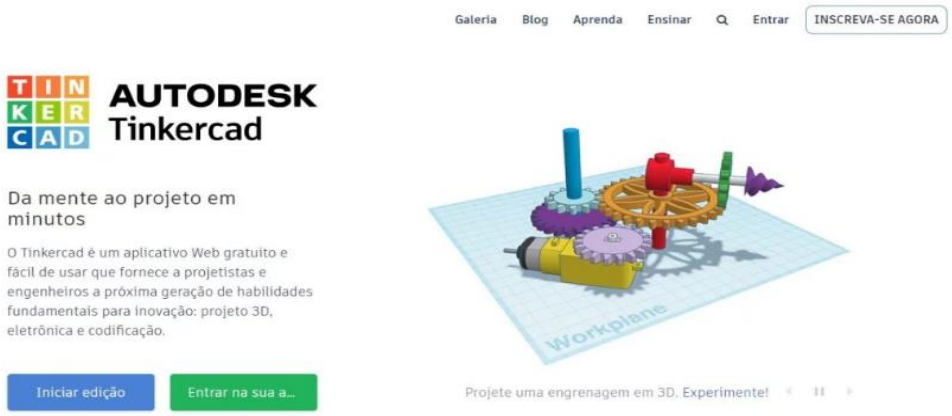
#include <LiquidCrystal.h>

7. Simulador *TinkerCad*

O *TinkerCad* é uma ferramenta online de design de modelos 3D em CAD, onde podemos criar projetos e enviar para uma impressora 3D por exemplo, e de simulação de circuitos elétricos analógicos e digitais, desenvolvida pela Autodesk. Por ser gratuita e de fácil manipulação, encontramos nela uma oportunidade de ensino de Programação Embarcada, que é o principal objetivo de nossa aula neste semestre.

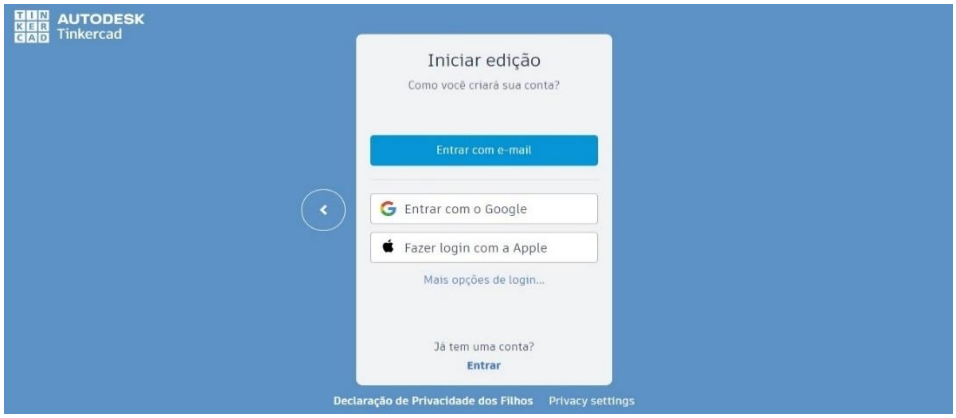
Através do link abaixo, você será redirecionado à página do TinkerCad: https://www.tinkercad.com/

Acessando-o, você deve se encontrar na página que está ilustrada na imagem abaixo:

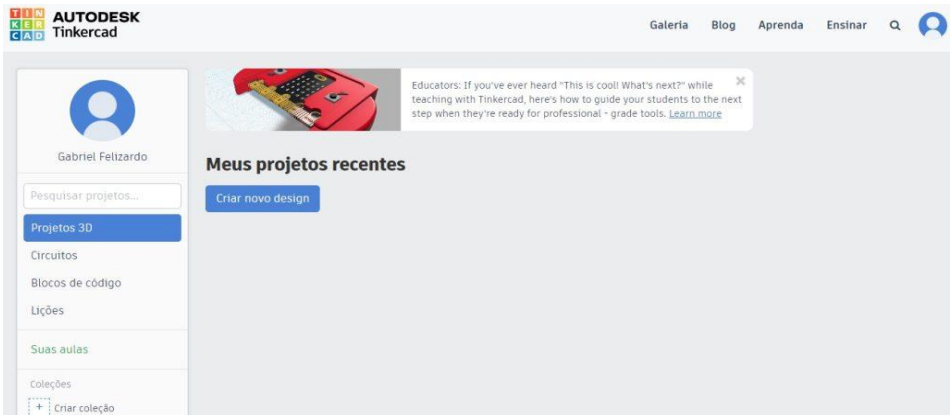


Perceba que, na imagem acima, temos dois botões no canto inferior esquerdo. “Iniciar a edição” lhe dará algumas opções para começar no TinkerCad, como educador, aluno ou por conta própria. Já o botão verde, “Entrar na sua aula”, irá te redirecionar para uma página que irá pedir um código de acesso para a sala de aula de um professor. Neste post, seguiremos, primeiro, pela opção “inscreva-se agora”.

Conforme vimos acima, iremos escolher a opção “Criar uma conta pessoal”.

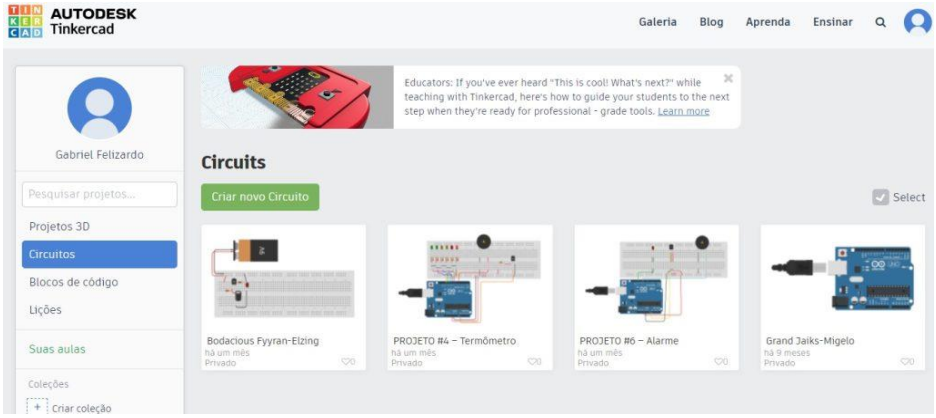


Agora, iremos criar a nossa conta pela opção “Entrar com o Google”. Depois deste passo, você só precisa preencher seus dados. Por fim, com todos estes passos concluídos, iremos ser direcionados para nosso dashboard. Você pode vê-lo logo abaixo:



Circuitos

Conforme a imagem abaixo, seguiremos pela aba “Circuitos” para começarmos a conhecer melhor esta ferramenta.



Após acessar “Criar novo circuito”, estaremos no ambiente de prototipação para nossos projetos em Arduino.

8. Programando o Arduino

Arduino se programa em uma linguagem de alto nível semelhante a C/C++ e geralmente tem os seguintes componentes para elaborar o algoritmo, isso tanto no simulador (TinkerCad) como na IDE do Arduino:

- Estruturas (veremos a seguir)
- Variáveis (veremos a seguir)
- Operadores booleanos, de comparação e aritméticos (no decorrer do curso)
- Estrutura de controle (no decorrer do curso)
- Funções digitais e analógicas (no decorrer do curso)

Veja a referência extendida (http://arduino.cc/en/Reference/HomePage?from=Reference.Extended) para características mais avançadas da linguagem Arduino e a página das bibliotecas

(http://arduino.cc/en/Reference/Libraries) para interação com tipos específicos de hardware, no site oficial do Arduino.

8.1. Estruturas

São duas funções principais que deve ter todo programa em Arduino.

A função setup() é chamada quando um programa começa a rodar. Use esta função para inicializar as sua variáveis, os modos dos pinos, declarar o uso de livrarias, etc. Esta função será executada apenas uma vez após a placa Arduino ser ligada ou ressetada.

```
setup(){

}
```

Após criar uma função setup() que declara os valores iniciais, a função loop() faz exatamente o que seu nome sugere, entra em looping (executa sempre o mesmo bloco de código), permitindo ao seu programa fazer mudanças e responder. Use esta função para controlar ativamente a placa Arduino.

```
loop(){

}
```

8.2. Variáveis

Variáveis são expressões que você pode usar em programas para armazenar valores como a leitura de um sensor em um pino analógico. Aqui destacamos algumas:

- Variáveis Booleanas

Variáveis boolenas, assim chamadas em homenagem a George Boole, podem ter apenas dois valores: verdadeiro (true) e falso (false).

Exemplo: boolean running = false;

- Int

Inteiro é o principal tipo de dado para armazenamento numérico capaz de guardar números de 2 bytes. Isto abrange a faixa de -32.768 a 32.767 (valor mínimo de -2^15 e valor máximo de (2^15) -1).

Exemplo: int ledPin = 13;

- Char

Um tipo de dado que ocupa 1 byte de memória e armazena o valor de um caractere ASCII. Caracteres literais são escritos entre aspas.

Exemplo: char myChar = ‘A’;

9. Vamos praticar?

A ideia agora é iniciarmos o manuseio do Arduino com a Linguagem de Programação, para isso vamos usar exemplos de aplicações envolvendo o hardware e o software onde teremos a entrada, o processamento e a saída de dados, lembrando que iremos fazer primeiramente no TinkerCad e posteriormente fisicamente utilizando a IDE do Arduino.

9.1. *Hello World*

Este exemplo mostra a experiência mais simples que você pode fazer com um Arduino para verificar uma saída física: **piscar um LED**.

Quando você está aprendendo a programar, na maioria das linguagens de programação, o primeiro código que você escreve diz “*Hello World*” na tela do computador. Como a placa Arduino não tem uma tela substituiremos esta função fazendo piscar um LED.

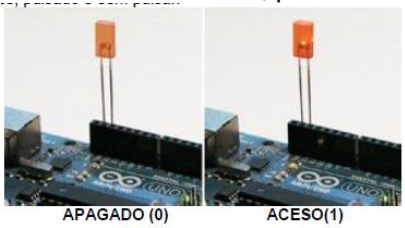
9.1.1. O que vou aprender?

- Ativar uma saída digital
- Acender um LED em ON/OFF
- Temporizar um sinal de saída
- Sintaxe de um programa Arduino

9.1.2. Conhecimentos prévios

9.1.2.1. Sinal digital

As entradas e saídas de um sistema eletrônico serão consideradas como sinais variáveis. Em eletrônica se trabalha com variáveis que são tomadas na forma de tensão ou corrente, que podem simplesmente ser chamados de sinais.
Os sinais podem ser de dois tipos: digital ou analógico.
Em nosso caso como o que nos interessa no momento é o sinal digital, ele se caracteriza por ter dois estados diferentes e portanto também podem ser chamadas de binárias (em lógica seria valores Verdadeiro (V) e Falso (F), ou poderiam ser 1 ou 0 respectivamente).
Um exemplo de um sinal digital é o interruptor da campainha da sua casa, porque ele tem somente dois estados, pulsado e sem pulsar.


APAGADO (0) ACESO(1)

9.1.2.2. Função pinMode()

Configura o pino especificado para que se comporte ou como uma entrada (input) ou uma saída (output).

Sintaxe:

pinMode(pin, mode)

pinMode(9, OUTPUT); // determina o pino digital 9 como uma saída.

9.1.2.3. Função digitalWrite()

Escreve um valor HIGH (ligar) ou um LOW (desligar) em um pino digital.

Sintaxe:

digitalWrite(pin, valor)

digitalWrite(9,HIGH) // o LED na porta 9 acenderá

9.1.2.4. Função delay()

É um temporizador, onde o valor passado(em milisegundos) será o tempo em que o Arduino não irá executar atividades até que este tempo passes.

Sintaxe:

delay(valor em milisegundos)

delay(1000); // neste caso o Aduino ficará 1 segundo sem executar atividades.

9.1.2.5. Polaridade de um LED

O LED (*Light Emitting Diode*) é um diodo que emite luz quando energizado. Os LED´s apresentam muitas vantagens sobre as fontes de luz incandescentes como um consumo menor de energia, maior tempo de vida, menor tamanho, grande durabilidade e confiabilidade. O LED tem uma polaridade, uma ordem de conexão. Ao conectá-lo invertido não funcionará corretamente. Revise os desenhos para verificar a correspondência do negativo e do positivo.

São especialmente utilizados em produtos de microeletrônica como sinalizador de avisos. Também é muito utilizado em painéis, cortinas e pistas de led. Podem ser encontrados em tamanho maior, como em alguns modelos de semáforos ou displays.

9.1.2.6. Conexão da placa Arduino com o computador

Item 6.3 desta nota de aula.

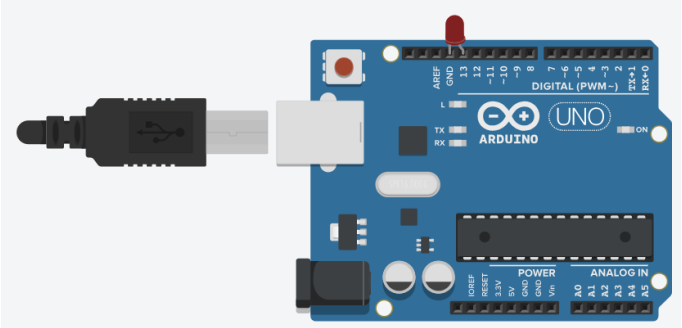
9.1.2.7. Material necessário

1 Arduino Uno

1 LED

1 Cabo USB AB

9.1.2.8. Diagrama



9.1.2.9. Código fonte

No programa a seguir, o primeiro comando é o de inicializar o pino 13 como saída através da linha pinMode(13, OUTPUT);

No loop principal do código, você liga o LED com esta linha de comando:

digitalWrite(13, HIGH);

Este comando direciona 5 volts ao pino 13 e o acende. Você desliga o LED com o seguinte comando:

digitalWrite(13, LOW);

Este comando retira os 5 volts do pino 13, voltando para 0 e desligando o LED. Entre desligar e ligar você precisa de tempo suficiente para que uma pessoa veja a diferença, então o comando delay() informa o Arduino não fazer nada durante 1000 milissegundos, ou um segundo. Quando você usa o comando delay(), nada mais acontece neste período de tempo.

```
/*
Piscar/Led -> "Hello World - Arduino"
Acende o Led por um segundo, e depois apaga pelo mesmo tempo,
de forma repetida.
*/

//Variável para o pino 13
int led = 13;

//Executa somente 1 vez quando o Arduino liga
void setup(){
    //Indica que o pino é de Saída
    pinMode(led, OUTPUT);
}

void loop(){
    digitalWrite(led, HIGH); //Acende o Led
    delay(1000); //Pausa de 1 segundo
    digitalWrite(led, LOW); //Apaga o Led
    delay(1000); //Pausa de 1 segundo
}
```

Agora façam primeiramente no TinkerCad, quando estiver funcionando corretamente, façam o mesmo projeto de forma física com os componentes que estarei disponibilizando no laboratório.

9.2. Semáforo

Agora nós iremos criar um circuito para simular um semáforo de trânsito. O semáforo será constituído por três LED's: um vermelho, um amarelo e um verde.

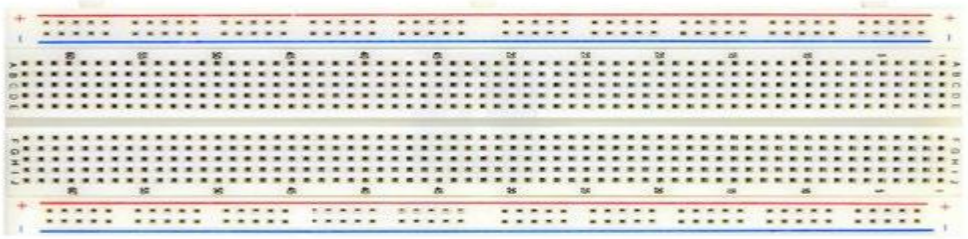
9.2.1. O que vou aprender?

- O que é uma protoboard;
- Cabear um circuito;
- Ler uma entrada digital e escrever uma saída digital.

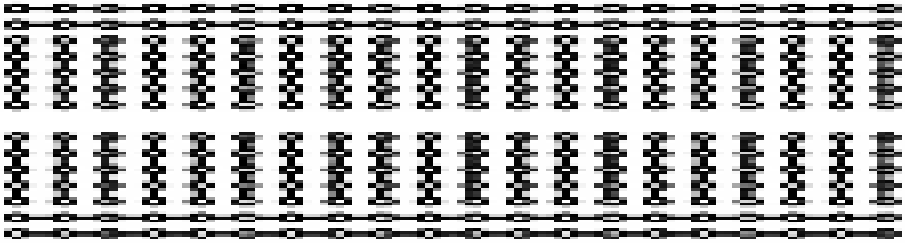
9.2.2. O que é uma Protoboard

É uma placa reutilizável usada para construir protótipos de circuitos eletrônicos sem solda.

Uma protoboard é feita por blocos de plástico perfurados e várias lâminas finas de uma liga metálica de cobre, estanho e fósforo.

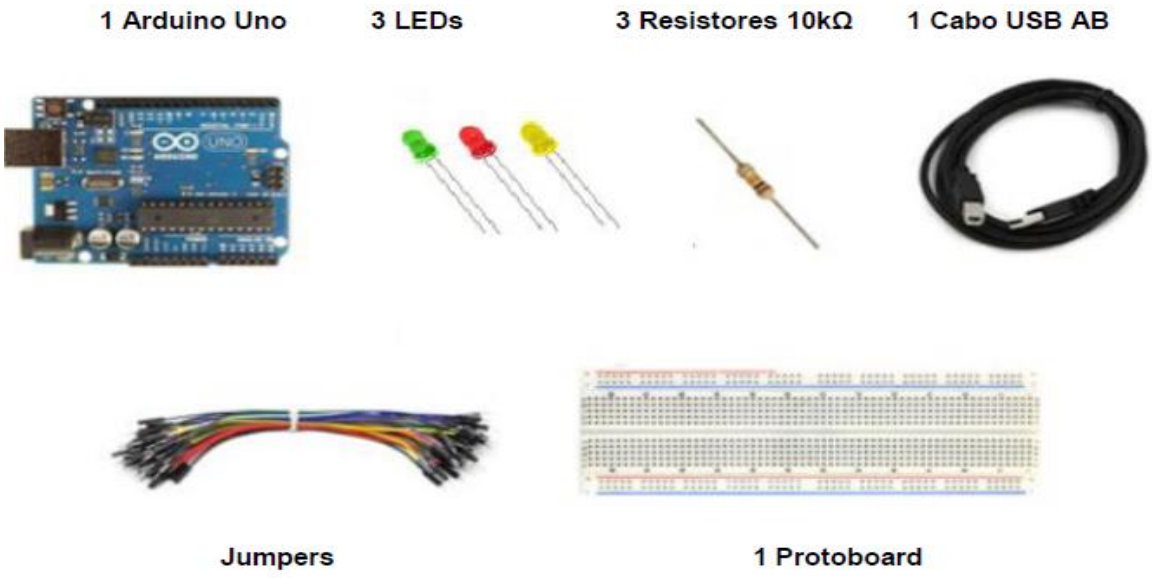


Protoboard

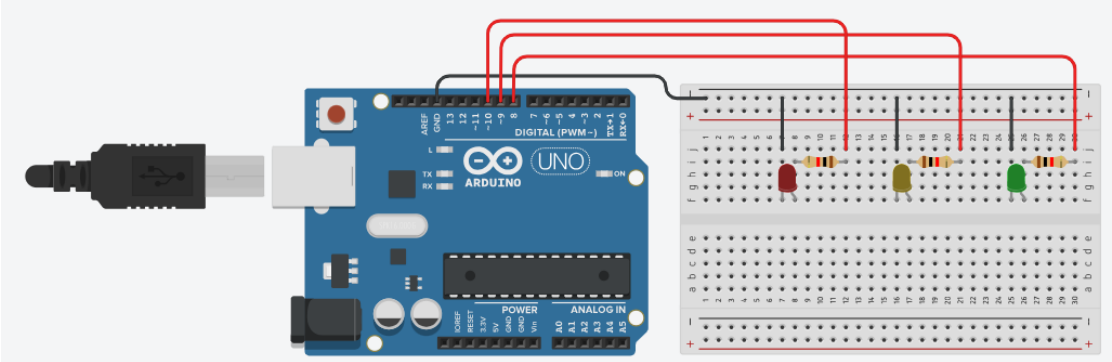


Conexões abertas

9.2.3. Material necessário



9.2.4. Diagrama



9.2.5. Código Fonte

```
/*
Semáforo simples.
*/
int ledDelay = 5000; //Variável de tempo das paradas

int vermelho = 10; //Variável de Luz Vermelha
int amarelo  = 9;  //Variável de Luz Amarela
int verde    = 8;  //Variável de Luz Verde

void setup(){
    //Inicializando os pinos
    pinMode(vermelho, OUTPUT);
    pinMode(amarelo, OUTPUT);
    pinMode(verde, OUTPUT);
}

void loop(){
    digitalWrite(vermelho, HIGH); //Acende o vermelho
    delay(ledDelay); //Aplicando a pausa de tempo
    digitalWrite(vermelho, LOW); //Apagando o Vermelho
    delay(500); //Pausa para transição
    digitalWrite(amarelo, HIGH); //Acende o Amarelo
    delay(2000); //Pausa de atenção
    digitalWrite(amarelo, LOW); //Apagando o Amarelo
    delay(500); //Pausa para transição
    digitalWrite(verde, HIGH); //Acende o Verde
    delay(ledDelay); //Aplicando o tempo que ficará verde
    digitalWrite(verde, LOW); //Desliga o Verde
    delay(500); //Pausa de transição
}
```

Agora façam primeiramente no TinkerCad, quando estiver funcionando corretamente, façam o mesmo projeto de forma física com os componentes que estarei disponibilizando no laboratório.

9.3. Botão

O botão é um componente que conecta dois pontos do circuito quando está pressionado. Neste exemplo quando o botão está pressionado o LED se acende.

9.3.1 O que vou aprender?

- Cabear um circuito;
- Condicional if/else;
- Comunicação serial;
- Função digitalRead(), ler uma entrada digital.

9.3.2. Conhecimentos prévios

9.3.2.1. Função digitalRead()

Lê o valor de um pino digital especificado, HIGH ou LOW.

Sintaxe:

digitalRead(pin)

buttonState = digitalRead(9); // Leitura do estado de um botão no pino 9.

9.3.2.2. Condicional If else (Se e Senão)

9.3.2.2.1. Condicional If else

If, que é usado juntamente com um operador de comparação, verifica quando uma condição é satisfeita, como por exemplo um input acima de um determinado valor. O formato para uma verificação **if** é:

```
if (algumaVariavel == 50) {
        // faça alguma coisa se verdade
} else{
        // faça outra coisa se falso
}
```

O programa checa se algumaVariavel (colocar acentos em nomes de variáveis não é uma boa ideia) é maior que 50. Se for, o programa realiza uma ação específica. Colocado de outra maneira, se a sentença que está dentro dos parêntesis é verdadeira o código que está

dentro das chaves roda; caso contrário o programa salta este bloco de código, e vai para o **else,** que é a negação do IF, realizando outra tarefa.

A sentença que está sendo verificada necessita o uso de pelo menos um dos operadores de comparação:

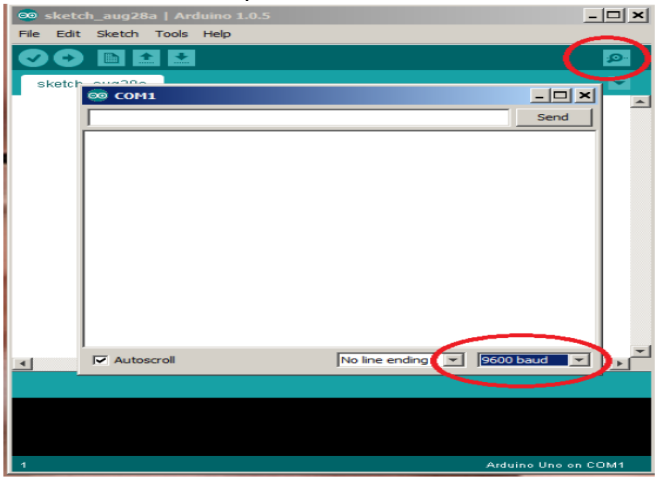
x == y (x é igual a y)
x != y (x é não igual a y)
x < y (x é menor que y)
x > y (x é maior que y)
x <= y (x é menor ou igual a y)
x >= y (x é maior ou igual a y)

9.3.3. *Serial.print*

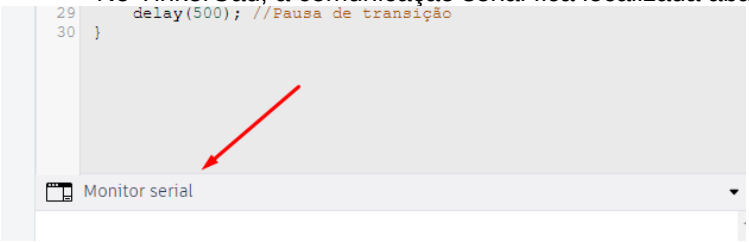
Exibe dados seriais sendo enviados da placa Arduino para o computador. Para enviar dados para a placa, digite o texto e clique no botão "enviar" ou pressione enter.

A comunicação entre a placa Arduino e seu computador pode acontecer em várias velocidades padrão pré-definidas. Para que isso ocorra é importante que seja definida a mesma velocidade tanto na *Sketch* quanto no Monitor Serial.

Na Sketch esta escolha é feita através da função Serial.begin. E no Monitor Serial através do menu *drop down* do canto inferior direito, isso na IDE Arduino.



No TinkerCad, a comunicação serial fica localizada abaixo do código escrito, vejam:

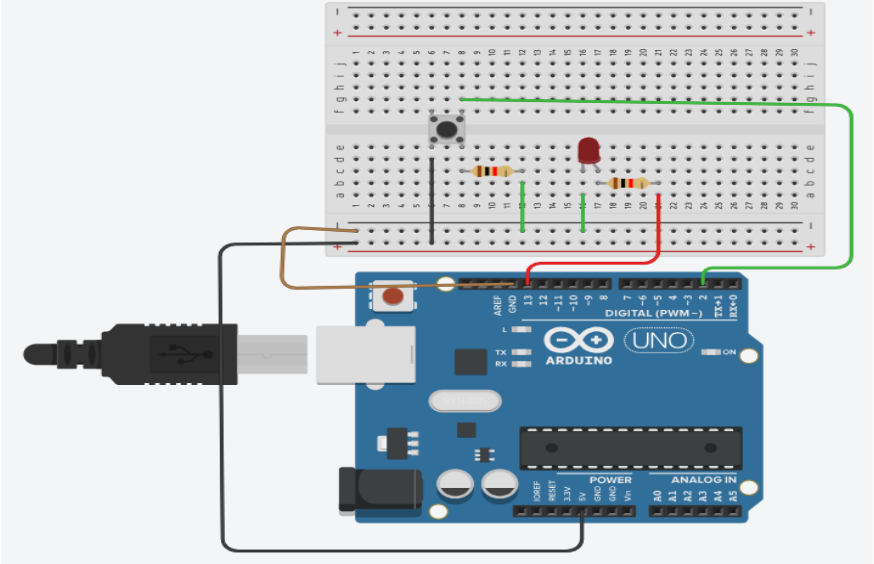


9.3.4. Material necessário

1 Arduino Uno **1 PushButton** **1 LED** **1 Resistor 10k** **Cabo USB/AB**

Jumpers **1 Protoboard**

9.3.5. Diagrama



9.3.6. Código Fonte

```
/*
   Botão para acionar LED
*/
//Declarar as variáveis de controle
int v_botao  = 2;
int v_ledRed = 13;
String v_situacao;

//Declarar a variável de estado do botão
int v_estado;

void setup() {
  Serial.begin(9600);
  pinMode(v_botao, INPUT);
  pinMode(v_ledRed, OUTPUT);
}

void loop() {
  //Leitura do estado do botão
  v_estado = digitalRead(v_botao);

  //Validação
  if (v_estado == HIGH){
    //Acender o LED
    digitalWrite(v_ledRed, HIGH);
    v_situacao = "LED ligado :)";
  }else{
    //Apagar o LED
    digitalWrite(v_ledRed, LOW);
    v_situacao = "LED desligado :(";
  }
  Serial.println(v_situacao);//Mostrando o resultado na Serial
  delay(1000);
}
```

Agora façam primeiramente no TinkerCad, quando estiver funcionando corretamente, façam o mesmo projeto de forma física com os componentes que estarei disponibilizando no laboratório.

9.4. Som no Arduino

Agora nós iremos ver como emitir sons com o Arduino e mais algumas definições importantes tanto em programação convencional como embarcada, iremos fazer uma escala musical, utilizando os recursos explicados a seguir.

9.4.1. O que vou aprender

- O que é um ***Buzzer***;
- Função ***tone()***;
- O que é um ***Array***;
- Estrutura de repetição ***for***.

9.4.2. O que é um *Buzzer*

O *buzzer* é um transdutor ou espécie de alto-falante que já possui um oscilador interno para produzir sons (semelhantes ao de uma sirene). Os transdutores são dispositivos de alta impedância, fabricados com um material à base de uma cerâmica, denominada titanato de bário. Eles são muito sensíveis podendo ser usados como fones ou pequenos alto-falantes em sistemas de aviso ou alarmes.

9.4.3. Função tone()

A função tone() possui 2 sintaxes: tone(pino, frequência) e tone(pino, frequência, duração), onde pino referencia qual é o pino que irá gerar a frequência (ligado ao positivo do buzzer), a frequência é definida em hertz e a duração (opcional) é em milissegundos. Caso opte pela sintaxe sem duração é necessário usar a função noTone(pino) para parar a frequência enviada pelo pino definido

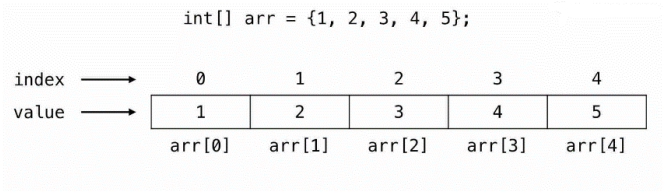
Exemplo:

tone(pino, frequencia, duração)

onde a **frequencia** do tom é setada em hertz, e a **duração**, em milisegundos.

9.4.4. O que é um *Array*?

Array é um tipo especial de variável que pode armazenar uma série de elementos de dados ao mesmo tempo. Além disso, cada valor de um *array* estará sempre relacionado à uma chave de identificação. Portanto, esta estrutura de dados é também conhecida como variável indexada, vejam a imagem abaixo:



9.4.5. Estrutura de repetição *for*

A estrutura *for* é usada para repetir um bloco de instruções entre chaves. Um contador de incremento é geralmente usado para incrementar e finalizar o loop. A instrução for é útil para qualquer operação repetitiva e geralmente é usada em combinação com matrizes para operar em coleções de dados / pinos, a seguir temos a sintaxe:

```
for (inicialização; condição; incremento) {
        // afirmações e código desejado para repetição;
}
```

Parâmetros:

- inicialização: acontece primeiro e exatamente uma vez;

- condição: cada vez que o loop é testado; se for verdade, o bloco de instruções e o incremento forem executados, a condição será testada novamente. Quando a condição se torna falsa, o loop termina;
- incremento: executado sempre através do loop quando a condição for verdadeira.

9.4.6. Material necessário

1 Arduino Uno

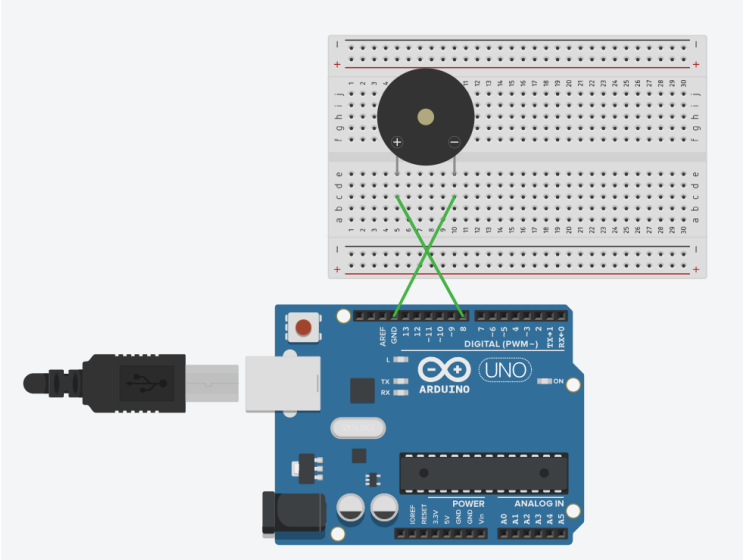
1 *Buzzer*

1 Cabo USB AB

Jumpers

1 *Protoboard*

9.4.7. Diagrama



9.4.8. Código fonte

```
/*
Buzzer - Notas musicais com buzzer
*/

//Notas Musicais
int DO   = 262;  //Nota Do
int RE   = 294;  //Nota Ré
int MI   = 330;  //Nota Mi
int FA   = 349;  //Nota Fa
int SOL  = 392;  //Nota Sol
int LA   = 440;  //Nota La
int SI   = 494;  //Nota Si
int DO_2 = 523;  //Nota Dó2

```

```
int pinoBuzzer = 8; //Pino do Buzzer

//Array para armazenar as notas
int melodia[] = {
  DO, RE, MI, FA, SOL, LA, SI, DO_2
// 0   1   2   3   4   5    6   7
};

int tempo[] = {
   300, 300, 500, 500, 200, 250, 600, 120
//  0    1    2    3    4    5    6    7
};

void setup() {
   // configura pino do buzzer como saída
  pinMode(pinoBuzzer,OUTPUT);
}

void loop() {
  for(int i=0 ; i<8; i++) //Estrutura for para percorrer o array
  {
    tone(pinoBuzzer, melodia[i]); //Toca de acordo com a nota
    delay(tempo[i]); //Duração do som
  }

  for(int i=7 ; i>=0; i--) //Estrutura for para percorrer o array
  {
    tone(pinoBuzzer, melodia[i]); //Toca de acordo com a nota
    delay(tempo[i]); //Duração do som
  }
}
```

Agora façam primeiramente no *TinkerCad*, quando estiver funcionando corretamente, façam o mesmo projeto de forma física com os componentes que estarei disponibilizando no laboratório.

9.5. Semáforo interativo com pedestre

Agora nós iremos criar um circuito para simular um semáforo de trânsito e que também tenha a interação humana (pedestre) acionando um botão para que o semáforo feche para ocorrer a travessia. O semáforo será constituído por três LED's: um vermelho, um amarelo e um verde para os carros, e dois para o pedestre, sendo um vermelho e outro verde.

9.5.1. O que vou aprender?

- Funções apartadas;
- Função ***millis()***;
- Conceito de constantes.

9.5.2. Conhecimentos prévios

9.5.2.1. Função *millis*()

Função que retorna o intervalo de tempo, em milisegundos, decorrido desde o início do programa em execução.

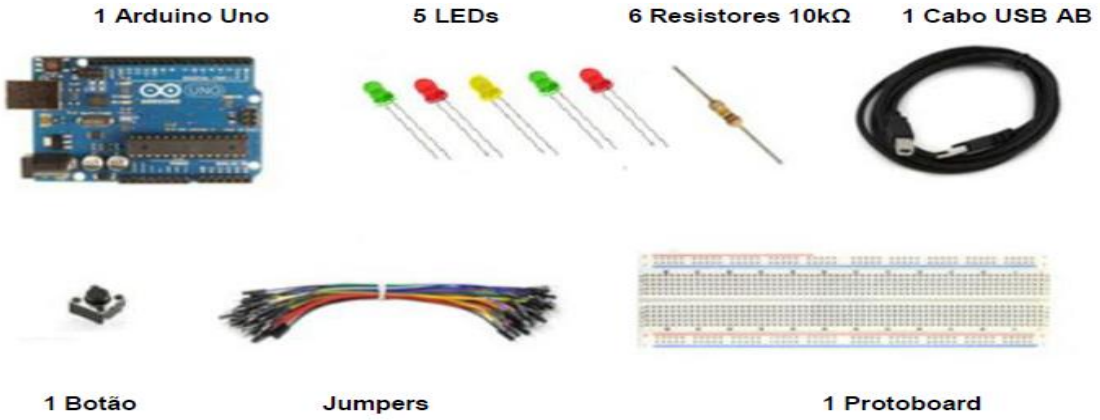
9.5.2.2. Funções

Uma função e simplesmente um bloco de código separado que recebeu um nome. Entretanto, funções também podem receber parâmetros e/ou retornar dados. Nesse caso, você não passou nenhum dado à função, nem fez com que ela retornasse dados. No caso de nosso exercício trataremos a função *changeLights().*

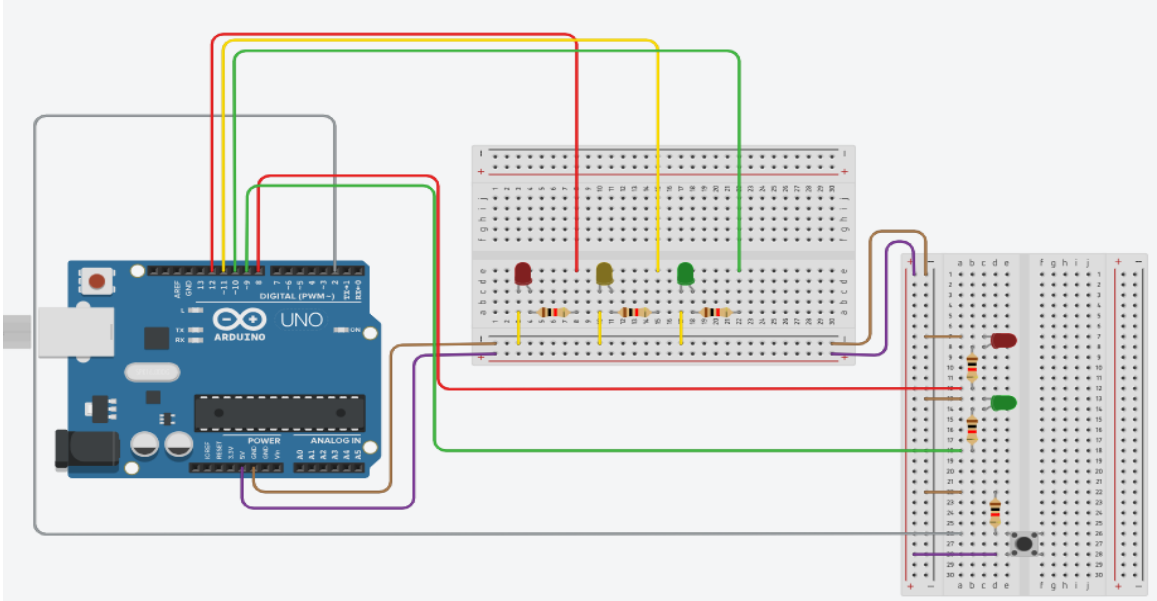
9.5.2.3. Constantes

Uma constante, nada mais é que um valor armazenado em memória que não tem seu valor alterado durante todo o ciclo de vida do algoritmo, recebido o valor, permanece com ele até o final de sua execução, e caso exista a tentativa de mudança desse valor, o programa exibirá um erro.

9.5.3. Material necessário



9.5.4. Diagrama



9.5.5. Código Fonte

```
/*
    Semáforo interativo com ação do pedestre
*/

//Constantes referentes aos semafóros
const int carroVermelho = 12;
const int carroAmarelo  = 11;
const int carroVerde    = 10;
const int pedVermelho   = 8;
const int pedVerde      = 9;
const int botao         = 2;

//Variáveis referentes a tempo e estado do semáforo
int tempoTravessia      = 5000;
int estado              = 0;
long changeTime;

void setup() {
  //Tipando as pinagens
  pinMode(carroVermelho, OUTPUT);
  pinMode(carroAmarelo, OUTPUT);
  pinMode(carroVerde, OUTPUT);
  pinMode(pedVermelho, OUTPUT);
  pinMode(pedVerde, OUTPUT);
  pinMode(botao, INPUT);

  //Iniciliazação da comunicação serial
  Serial.begin(9600);
}

```

```
void loop(){
  int state = digitalRead(botao);

  digitalWrite(carroVerde, HIGH);
  digitalWrite(pedVermelho, HIGH);

  if (state == HIGH && (millis() - changeTime) > 5000){
    Serial.println("AGUARDE! Fechando semaforo para os carros.");
    changeLights();
    estado = 0;
  }else{
    if (estado == 0){
        Serial.println("Semaforo aberto para os carros e fechado para pedestre");
        estado = 1;
    }
  }

}

void changeLights() {
  digitalWrite(carroVerde, LOW);
  digitalWrite(carroAmarelo, HIGH);
  delay(2000);
  digitalWrite(carroAmarelo, LOW);
  digitalWrite(carroVermelho, HIGH);
  delay(1000);
  digitalWrite(pedVermelho, LOW);
  digitalWrite(pedVerde, HIGH);
  Serial.println("Pedestre, pode atravessar!");
  delay(tempoTravessia);

  for(int x = 10; x >= 1; x--){
    Serial.println("ATENCAO, o semaforo do pedestre fechara em " + (String)x + " segundos");
    digitalWrite (pedVerde, HIGH);
    delay(500);
    digitalWrite (pedVerde, LOW);
    delay(500);
  }

  digitalWrite(pedVermelho, HIGH);
  delay(500);
  digitalWrite(carroVermelho, LOW);
  delay(1000);
  digitalWrite(carroVerde, HIGH);
  changeTime = millis();
}
```

Agora façam primeiramente no *TinkerCad*, quando estiver funcionando corretamente, façam o mesmo projeto de forma física com os componentes que estarei disponibilizando no laboratório.

9.6 Medindo distância com sensor ultrassônico

Neste projeto, iremos fazer um sensor que mede a distância de um determinado objeto e veremos o resultado na Serial Monitor.

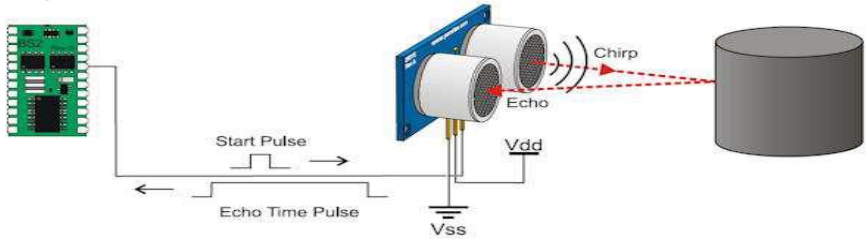
A comunicação seria no computador é vista em uma tela à parte, que pode ser acessada pelo atalho Ctrl+Shift+M (Serial monitor) na IDE do Arduino.

9.6.1 O que vou aprender?

- O que é um sensor ultrassônico;

9.6.2 O que é um sensor ultrassônico?

O sensor é usado para medir distâncias. A distância do sensor de ultrassom comum de medição é de cerca de 2 cm a 3-5m. O módulo sensor funciona assim:



O sensor ultrassônico é composto de um emissor e um receptor de ondas sonoras. Podemos compará-los a um alto-falante e um microfone trabalhando em conjunto. Entretanto, ambos trabalham com ondas de altíssima frequência, na faixa dos 40.000 Hz (ou 40KHz). Isto é

muito, muito acima do que os nossos ouvidos são capazes de perceber. O ouvido humano consegue, normalmente, perceber ondas na entre 20 e 20.000 Hz e por isto o sinal emitido pelo sensor ultrassônico passa despercebido por nós.

O sinal emitido, ao colidir com qualquer obstáculo, é refletido de volta na direção do sensor. Durante todo o processo, o aparelho está com uma espécie de “cronômetro” de alta precisão funcionando. Assim, podemos saber quanto tempo o sinal levou desde a sua emissão até o seu retorno. Como a velocidade do som no ar é conhecida, é possível, de posse do tempo que o sinal levou para ir até o obstáculo e voltar, calcular a distância entre o sensor e o obstáculo. Para isto vamos considerar a velocidade do som no ar (340 m/s) na seguinte equação:

$$d = (V * t) /2$$

Onde:

d = Distância entre o sensor e o obstáculo (é o que queremos descobrir);

V = Velocidade do som no ar (340 m/s);

t = Tempo necessário para o sinal ir do sensor até o obstáculo e voltar (é o que o nosso módulo sensor ultrassom mede);

A divisão por dois existe pois o tempo medido pelo sensor é na realidade o tempo para ir e voltar, ou seja, duas vezes a distância que queremos descobrir.

O funcionamento do sensor ultrassônico trabalha com dois pinos que são utilizados para alimentar o sensor, um deles é utilizado para disparar o sinal ultrassônico e o outro para medir o tempo que ele leva para retornar ao sensor. Existem alguns sensores ultrassônicos, que possuem apenas três pinos e utilizam um único pino para disparar o pulso e medir o tempo de resposta. Se seu sensor tiver apenas três pinos as conexões e o código do Arduino devem ser devidamente ajustados.

VCC: Alimentação do módulo com +5 V.

Trig: Gatilho para disparar o pulso ultrassônico. Para disparar coloque o pino é HIGH por pelo menos 10us.

Echo: Gera um pulso com a duração do tempo necessário para o eco do pulso ser recebido pelo sensor.

Gnd: Terra.

Vale lembrar que existem limitações para o funcionamento do sensor ultrassônico. Primeiro você tem que levar em consideração que tipo de obstáculo está querendo detectar. Se o obstáculo for muito pequeno pode ser que ele não gere um sinal de retorno suficiente para ser percebido pelo sensor. Se o obstáculo não estiver posicionado bem a frente do sensor você pode ter medidas imprecisas ou até mesmo não acusar a presença do mesmo. E por fim, a faixa de distância que o sensor trabalha fica entre 2cm e 3-5m e isto pode variar de sensor para sensor.

9.6.4 Material necessário

1 Arduino Uno

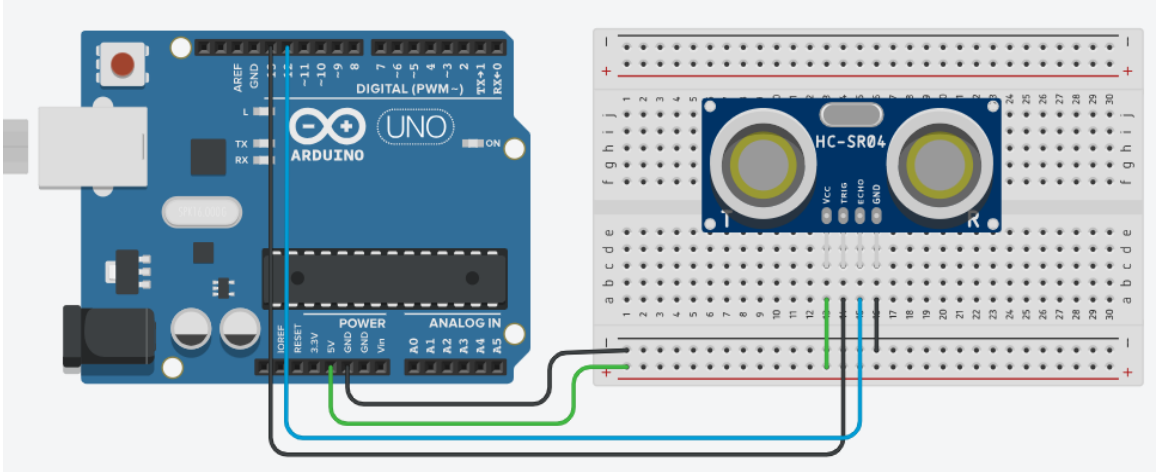
1 sensor ultrassônico

1 Cabo USB AB

Jumpers

1 Protoboard

9.6.5 Diagrama



9.6.6 Código Fonte

Vocês verão a distância calculada pelo código no Monitor Serial. Mas como foi calculada essa distância?

Anteriormente, falamos que o sensor emite pulsos de ondas. Esses pulsos são enviados pelo pino Trigger a cada 10us (microssegundos) e retornam para o Echo. A onda ultrassônica percorre o ar em uma velocidade de 343,5 m/s. Portanto, o cálculo feito para encontrar a distância medida é:

(343,5 m/s * 100 cm)/(1.000.000) = 0,3435

0,3435/2 = 0,017175

Multiplicamos a velocidade por 100 para a leitura ser feita em centímetros e após isso dividimos por 1 milhão (equivalente aos 10us). Esse valor é o resultado da distância de ida mais a da volta da onda, por isso, ainda é preciso dividi-lo por 2. Assim, encontramos a variável distância, que é igual a duração do pulso vezes o valor 0,017175.

```
/*
    Trabalhando com sensor ultrassônico
*/

const int PinoTrigger = 13;    //O Trigger emite o pulso
const int PinoEcho = 12;       //O Echo recebe o pulso

int duracao;    //Armazenamento do valor lido
int distancia; //Distância calculada

void setup () {
    pinMode(PinoTrigger,OUTPUT);
    pinMode(PinoEcho,INPUT);
    Serial.begin(9600);
}

void loop(){
    digitalWrite(PinoTrigger, HIGH);
    delayMicroseconds(10);
    digitalWrite(PinoTrigger,LOW);

    duracao = pulseIn(PinoEcho, HIGH); //Armazena o valor lido
    distancia = duracao*0.017175; //Calculo de tempo em distância
    Serial.print(distancia);
    Serial.println("cm");
    delay(100);
}
```

9.7. Motor DC e Ponte H

Este exemplo fará a movimentação de dois motores DC, com a utilização da Ponte H para podermos inverter o sentido do motor, quando necessário.

9.7.1. O que vou aprender?

- O que é um motor DC.
- O que é uma ponte H – L293D

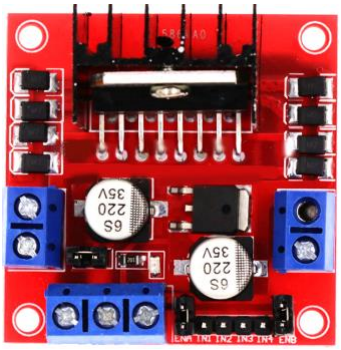
9.7.2. Conhecimentos prévios

9.7.2.1. Motor DC

Os motores de corrente contínua (CC) ou motores DC (*Direct Current*), como também são chamados, são dispositivos que transformam energia elétrica em mecânica aproveitando as forças de atração e repulsão geradas por eletroímãs e ímãs permanentes. Eles são comumente usados em projetos de robótica e automação devido à sua simplicidade e facilidade de controle. Para controlar um motor DC com o Arduino, é necessário usar um driver de motor, como o L298N, ou a famosa ponte H.

9.7.2.2. Ponte H – L293D

O chip L293D é reconhecido como um tipo de Ponte H, que consiste em um circuito elétrico usado para aplicar tensão a uma carga em qualquer direção de saída, como em nosso exemplo, um motor, que fará o motor ir para frente ou para trás. Este Módulo L293D é compatível tanto com o Arduino Uno.



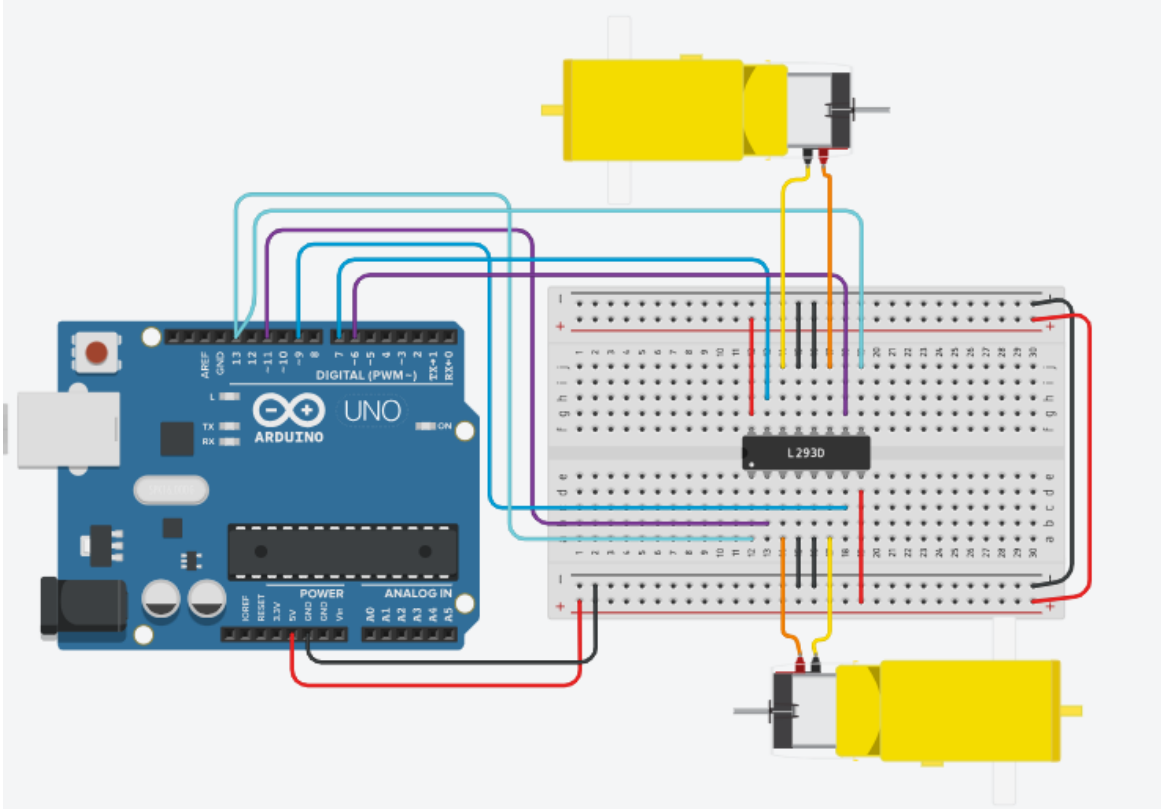
9.7.3. Material necessário

1 Arduino Uno **1 ponte H** **2 motores DC** **1 Cabo USB AB**

Jumpers

1 Protoboard

9.7.4. Diagrama



9.7.5. Código fonte

```
/*
    Professor Marcos Costa
    Acionamento de dois motores com ponte H L293D
*/


//Contantes da ponte H que ligam o motor
const int acti=13;
const int in1=11;
const int in2=9;
const int in3=6;
const int in4=7;

void setup()
{
  //Configurando os pinos
  pinMode(acti, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);

  //Acionando a ponte H
  digitalWrite(acti, HIGH);
}
```

```
26
27  void loop ()
28  {
29    //Movimento para frente
30    digitalWrite(in1, HIGH);
31    digitalWrite(in2, LOW);
32    digitalWrite(in3, HIGH);
33    digitalWrite(in4, LOW);
34    delay(2000);
35
36    //Parando os motores
37    digitalWrite(in1, LOW);
38    digitalWrite(in2, LOW);
39    digitalWrite(in3, LOW);
40    digitalWrite(in4, LOW);
41    delay(2000);
42
43    //Movimento para trás
44    digitalWrite(in1, LOW);
45    digitalWrite(in2, HIGH);
46    digitalWrite(in3, LOW);
47    digitalWrite(in4, HIGH);
48    delay(2000);
49
50    //Parando os motores novamente
51    digitalWrite(in1, HIGH);
52    digitalWrite(in2, HIGH);
53    digitalWrite(in3, HIGH);
54    digitalWrite(in4, HIGH);
55    delay(2000);
56
57  }
```