Instituto Federal de Educação Ciência e Tecnologia de Mato Grosso - Campus Cuiabá

Aprendendo a Programar em Arduino

Realização: PET Auto Net

Autor: Micael Bronzatti Gaier

2011

PET AutoNet

PROGRAMA DE EDUCAÇÃO TUTORIAL
PET

IFMT
Campus Cuiabá

Departamento da Área de Informática

Micael Bronzatti Gaier

PET AutoNet

Sumário

Introdução

A cada dia que passa a plataforma Arduino vem conquistando novos usuários. Tal sucesso é devido a sua simplicidade e ao fato de não necessitar conhecer profundamente a eletrônica e as estruturas de linguagens para criar gadgets, robôs ou pequenos sistemas inteligentes.

Devido ao aumento de procura dos estudantes do IFMT para realizar seus projetos com a plataforma Arduino, o Grupo PET AutoNet juntamente com o Departamento da Área de Eletro-Eletrônica (DAE-E), o Departamento da Área de Informática (DAI) e o Departamento de Pesquisa e Pós-Graduação (DPPG) desta instituição resolveram ministrar sob orientação do Prof. Dr. Ronan Marcelo Martins um minicurso sobre a plataforma Arduino aos estudantes interessados, de onde surgiu esta apostila.

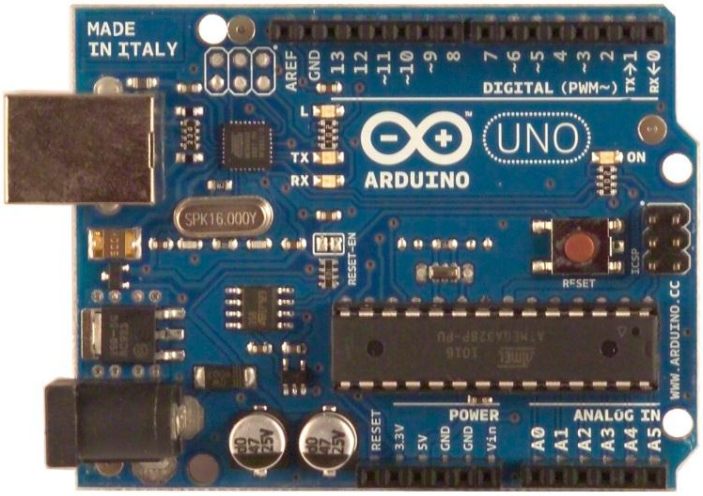
Esta apostila foi desenvolvida para complementar o aprendizado sobre Arduino, que possui tanto a parte do hardware como do software do Arduino proporcionando aos estudantes um melhor entendimento sobre o assunto. Esta apostila serve como apoio tanto para a comunidade interna ou externa ao IFMT devido às ações sociais realizadas pelo grupo PET AutoNet.

Conhecendo a Plataforma Arduino

O Arduino é uma ferramenta “open source” que vem sendo muito utilizado no meio acadêmico para a realização e desenvolvimento de diferentes projetos.

O Arduino é tanto um hardware como um software.

- **Hardware:** como hardware, o Arduino é uma plataforma de desenvolvimento em cima de microcontroladores da família Atmel, onde, invés de o responsável por um projeto em específico comprar diversos componentes e montar a sua própria placa, basta ele economizar tempo e dinheiro adquirindo uma plataforma simples e já pronta para programação. Há placas com diversas capacidades e variados microcontroladores, assim, podemos realizar diferentes tipos de projetos usando uma mesma plataforma de desenvolvimento.
  Neste minicurso, estaremos usando a plataforma **ARDUINO UNO.**



• • •

Texto extraído da Revista Saber Eletrônica; Ano 47; N° 454 – 2011; Editora Saber LTDA Páginas 12 a 15. Autor Filipe Pereira

Apresentação do Arduino: É uma placa microcontroladora baseada no microcontrolador ATmega 328. Ela tem 14 pinos I/O digitais, 6 entradas analógicas, um oscilador de 16 MHz (a cristal), uma conexão USB, um jaque de alimentação, um header ICSP, e um botão de reset. Ela contém tudo o que é necessário para o suporte do µC (microcontrolador), ligando-a simplesmente a um computador através de um cabo USB, ou alimentando-a com um adaptador AC/AD (ou bateria) para dar a partida.

• • •

• • •

Microcontrolador	ATmega328
Tensão de Funcionamento	5 V
Tensão de Entrada (recomendado)	7-12 V
Tensão de Entrada (limites)	6-20 V
Pinos E/S digitais	14 (dos quais 6 são saídas PWM)
Pinos de Entrada Analógicos	6
Corrente DC por pino E/S	40 mA
Corrente DC por pino 3.3V	50 mA
Memória Flash	32 KB, sendo 2KB utilizado pelo bootloader
SRAM	2 KB
EEPROM	1 KB

Alimentação

O Arduino pode ser alimentado pela ligação USB ou por qualquer fonte de alimentação externa. A fonte de alimentação é selecionada automaticamente. Alimentação externa (não USB) pode ser tanto de uma fonte como de uma bateria. A fonte pode ser ligada com um plugue de 2,1 mm (centro positivo) no conector de alimentação. Cabos vindos de uma bateria podem ser inseridos nos pinos GND (massa) e Vin (entrada de tensão) do conector de alimentação.

A placa pode funcionar com uma alimentação externa de 6 a 20 volts. Entretanto, se a alimentação for inferior a 7 volts, o pino 5 V pode fornecer menos de 5 volts e a placa poderá ficar instável. Se a alimentação for superior a 12 volts, o regulador de tensão poderá sobreaquecer e avariar a placa. A alimentação recomendada é de 7 a 12 volts.

Os pinos de alimentação são:

VIN - Entrada de alimentação para a placa Arduino quando uma fonte externa for utilizada. Poder-se-á fornecer alimentação por este pino ou, se for usar o conector de alimentação, empregar a alimentação por este pino.

5V - A fonte de alimentação utilizada para o microcontrolador e para outros componentes da placa pode ser proveniente do pino Vin através de um regulador on-board, ou ser fornecida pelo USB ou outra fonte de 5 volts.

3V3 - Alimentação de 3,3 volts fornecida pelo chip FTDI. A corrente máxima é de 50 mA.

GND - Pino terra ou massa.

• • •

• • •

Memória

O ATmega328 tem 32 KB de memória flash para armazenar código (dos quais 2 KB são utilizados pelo bootloader), além de 2 KB de SRAM e 1 KB de EEPROM (Electrically-Erasable Programmable Read-Only Memory), que pode ser lida e escrita através da biblioteca EEPROM.

Entrada e Saída

Cada um dos 14 pinos digitais do Arduino pode ser utilizado como entrada ou saída, usando as funções de pinMode(), digitalWrite(), e digitalRead(). Eles trabalham com 5 volts. Cada pino pode fornecer ou receber um máximo de 40 mA e tem uma resistência de pull-up interna (vem desligada de fábrica) de 20-50 kΩ. Além disso, alguns pinos têm funções específicas, a saber:

Serial: 0 (RX) e 1 (TX): são usados para receber (RX) e transmitir (TX) dados série, TTL. Estes pinos são ligados aos pinos correspondentes do chip serial FTDI USB-to-TTL

External Interrupts: 2 and 3 - Estes pinos podem ser configurados para ativar uma interrupção por um baixo valor, uma elevação ou falling edge ou uma mudança de valor. Veja a função attachInterrupt() para mais pormenores.

PWM: 3, 5, 6, 9, 10, e 11 - Fornecem uma saída analógica PWM de 8-bit com a função analogWrite().

SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK) - Estes pinos suportam comunicação SPI, que embora compatível com o hardware, não está incluída na linguagem do Arduino.

LED: 13 - Há um LED já montado e ligado de origem ao pino digital 13. Quando o pino está no valor HIGH, o LED acende; quando o valor está em LOW, ele apaga. O Arduino tem 6 entradas analógicas e cada uma delas tem uma resolução de 10 bits (i.e. 1024 valores diferentes). Por padrão, elas medem de 0 a 5 volts, embora seja possível mudar o limite superior usando o pino AREF e um pouco de código de baixo nível. Adicionalmente alguns pinos têm funcionalidades específicas, a saber:

I^2C: 4 (SDA) and 5 (SCL) - Suportam comunicação I^2C (TWI) usando a biblioteca.

Há ainda mais alguns pinos na placa:

AREF - Referência de tensão para entradas analógicas. São usados com o analogReference().

Reset - Envia o valor LOW para efetuar o RESET ao microcontrolador. É tipicamente utilizado para adicionar um botão de reset aos shields que bloqueiam o que há na placa.

• • •

• • •

É ainda importante referir que a corrente máxima por cada pino analógico e digital é de 40 mA, com exceção da saída que providencia 3,3 V, que permite correntes máximas de 50 mA.

De acordo com Sousa and Lavinia, 2006, a capacidade de utilizar Pulse Width Modulation (PWM) é muito importante, pois permite obter uma tensão analógica a partir de um sinal digital, ou seja, de um sinal que apenas pode assumir o estado lógico 0 (0 V) ou 1 (5 V). O conceito de PWM é usado para referir sinal que possua uma frequência constante e um "duty-cicle" variável.

Comunicação

Com o Arduino, a comunicação com um computador, com outro Arduino ou com outros microcontroladores é muito simplificada. O ATmega328 permite comunicação série no padrão UART TTL (5V), que está disponível nos pinos digitais 0 (RX) e 1 (TX), vide figura 2. Um chip FTDI FT232RL na placa encaminha esta comunicação série através do USB e os drives FTDI (incluído no software do Arduino) fornecem uma porta COM virtual para o software no computador. O software Arduino inclui um monitor série que permite que dados simples de texto sejam enviados à placa Arduino.

Os LEDs RX e TX da placa piscam quando os dados estão para ser transferidos para o computador pelo chip FTDI. Pela ligação USB não dá quando há comunicação série pelos pinos 0 e 1.

A biblioteca SoftwareSerial permite a comunicação série por quaisquer dos pinos digitais do Duemilanove.

O ATmega328 também oferece suporte aos padrões de comunicação I^2C (TWI) e SPI. O software do Arduino inclui uma biblioteca Wire para simplificar o uso do bus I^2C; para usar a comunicação SPI, veja a folha de informações do ATmega328.

Programação

O Arduino pode ser programado com o software Arduino (download).

O ATmega328 no Arduino vem pré-gravado com um bootloader que permite enviar novos programas sem o uso de um programador de hardware externo. Ele se comunica utilizando o protocolo original STK500 (referência, C header files). Também poder-se-á programar o ATmega328 através do ICSP (In-Circuit Serial Programming) header;

• • •

• • •

Reset automático (Software)

Algumas versões anteriores do Arduino requerem um reset físico (pressionando o botão de reset na placa) antes de carregar um sketch. Este Arduino é projetado de modo a permitir que isto seja feito através do software que esteja correndo no computador a que está ligado. Uma das linhas de controle de hardware (DTR) do FT232RL está ligada ao reset do ATmega328 por via de um condensador de 100 nF.

Quando é feito reset a esta linha (ativo baixo), o sinal cai por tempo suficiente para efetuar o reset ao chip. O software Arduino usa esta característica para permitir carregar o programa simplesmente pressionando-se o botão “upload” no ambiente Arduino. Isto significa que o “bootloader” pode ter um “timeout” mais curto, já que a ativação do DTR (sinal baixo) pode ser bem coordenada com o início do “upload”.

Considerando que é programado para ignorar dados espúrios (i.e. qualquer coisa a não ser um “upload” de um novo código), ele interceptará os primeiros bytes dos dados que são enviados para a placa depois que a ligação for aberta. Se um “sketch” rodando na placa receber uma configuração de uma vez, ou outros dados ao inicializar, dever-se-á assegurar que o software está em comunicação e espere um segundo depois de aberta a ligação, antes de enviar estes dados.

Proteção contra sobrecorrente USB

O Arduino tem um fusível que protege a porta USB do seu computador contra curto-circuito. Apesar da maioria dos computadores possuírem proteção interna própria, o fusível proporciona uma proteção extra. Se mais de 500 mA foram aplicados na porta USB, o fusível irá automaticamente interromper a ligação até que o curto ou a sobrecarga seja eliminada.

Características físicas

O comprimento e largura máximos do são 2,7” (68,50 mm) e 2,1” (53,34 mm) respectivamente, com o conector USB e o jack de alimentação indo um pouco além destas dimensões. Três furos de fixação permitem a montagem da placa numa superfície ou caixa. Note que a distância entre os pinos de entrada e saídas digitais nº 7 e nº 8 é de 160 mil (milésimos de polegada), não é sequer múltiplo do espaçamento de 100 mil dos outros pinos.

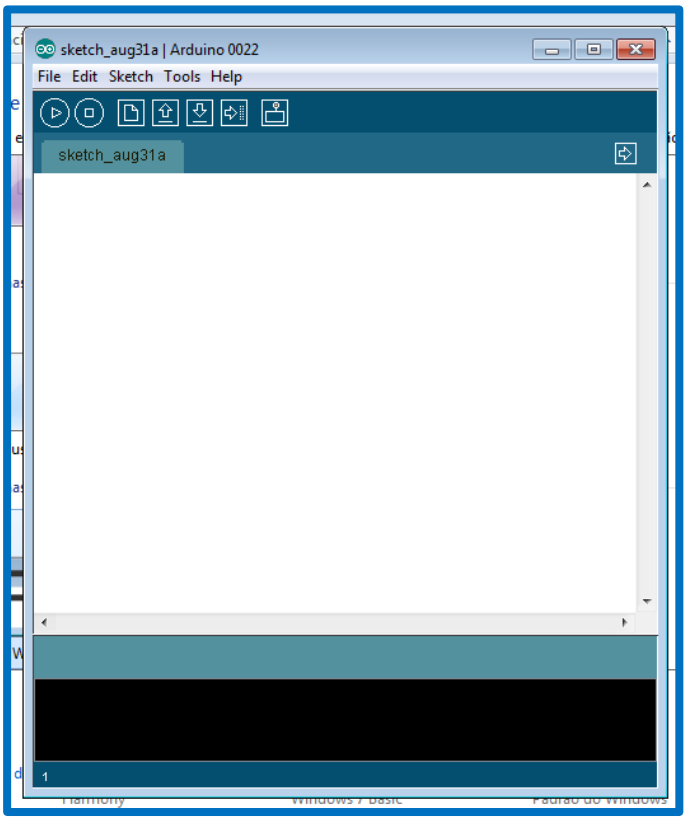
• • •

- **Software:** como Software, no Arduino é feita toda a parte de programação que será usada para controlar o hardware. A programação é baseada em linguagem C, porém, de forma mais simples. O software está disponível para Windows, Linux e Mac OS no site **www.arduino.cc**.

IDENTIFICAÇÃO DO HARDWARE EM WINDOWS 7.

Quando você for compilar algum programa e for passar para o hardware será necessário plugar o Arduino na entrada USB do seu computador, porém isso não se procede de forma automática e é necessário realizar o seguinte procedimento.

1. Conecte o cabo USB do Arduino no seu computador e aguarde o aviso de "Novo Hardware Encontrado";
2. Abra o Gerenciador de Dispositivos, em "Outros Dispositivos", aparecerá o hardware, então clique com o botão direito do mouse sobre o dispositivo e clique em "Atualizar/Instalar Hardware".
3. Na janela que aparecerá, clique em "Procurar Software do Driver no Computador";
4. Em seguida, você deverá identificar a pasta **exata** onde está localizado o drive do Arduino, ou seja, dentro da pasta do programa que você baixou do site haverá uma pasta chamada "Drivers", você deverá direcionar para esta pasta. Clique em avançar e espere a identificação do Arduino.



Software IDE do Arduino.

Base da Programação em Arduino

A programação em Arduino tem como base a Linguagem C. Para aprendermos a programar em Arduino, devemos aprender alguns conceitos básicos sobre Linguagem C.

OBSERVAÇÂO!

*Na Linguagem C, letras maiúsculas, minúsculas e conjuntos de palavras fazem a diferença. Ou seja, se você for escrever algo como, por exemplo, "**ledPin**", não será a mesma coisa que "**LedPin**".*

*O programa precisa identificar qual é o fim de uma linha de programação para poder seguir rodando o programa, para isso, é necessário ao final de cada linha onde possa ser identificado um comando, o uso de **;** (ponto e vírgula).*

Para demonstrar essa base, vamos pegar um simples programa, que se encontra como exemplo dentro do próprio software, sendo ele para ligar um pequeno LED que se encontra na placa do circuito, identificado como pino 13. ***EXEMPLO 1:***

```
/*
Liga o LED por um segundo e então o apaga por um
segundo, repentidamente.
Este exemplo de código é livre, de domínio publico.
Traduzido por Micael Bronzatti Gaier
*/

void setup() {
// inicializa o pino digital como saída de dados.
pinMode(13, OUTPUT);
}

void loop() {
digitalWrite(13, HIGH);    // liga o LED
delay(1000);               // espera por um segundo
digitalWrite(13, LOW);     // apaga o LED
delay(1000);               // espera por um segundo
}
```

COMENTÁRIOS

Quando desejamos realizar alguma instrução ou comentário com mais de uma linha sobre o programa, usamos como símbolo ***/**** ao início de um comentário e ****/*** ao término do mesmo.

Quando se deseja dar uma instrução breve, pode se usar ***//***, assim, quando você iniciar uma nova linha, voltará a editar o programa novamente.

FUNÇÕES - BASE

Um programa possui um corpo, uma função principal, onde o programa será rodado.

Em Arduino, é obrigatório que o programa tenha a função *void* ***setup()*** e a função *void* ***loop()***. Caso seu programa não tenha estas funções, ele não será capaz de identificar a linguagem e não realizará a compilação.

A função *void* ***setup()*** funciona como uma "*função de inicialização*", o que estará contido nela rodará apenas uma vez no programa. Nela, deve conter as informações principais para iniciar o seu programa, que estará contido dentro da função *void* ***loop()*** . Em *void* ***loop()*** deverá conter tudo o que você quiser que o programa faça.

pinMode(pino, MODO)

No Arduino, como lidamos com entrada e saída de dados, o *pinMode* configura um pino específico da placa como entrada ou saída. O **pino** é um número inteiro e ele é identificado na própria placa. O **MODO** ele pode ser de entrada (***INPUT***) ou saída (***OUTPUT***) de dados.

OBSERVAÇÃO:

Quando se tratar de uma leitura de dados analógicos, no caso os pinos ***A0*** *ao* ***A5****, não é necessário realizar a declaração destes pinos em void* ***setup()****, porém, os pinos analógicos, também podem ser usados como pinos digitais, assim, quando estiver se referindo ao uso destes pinos como digitais, você deverá* ***identificar no programa*** *na* ***função de inicialização*** *como pino* ***14*** *(no caso do* ***A0****) até o* ***19*** *(no caso do* ***A5****).*

ENTRADA E SAÍDA DE DADOS I

Quando desejarmos introduzir dados ou ler dados recebidos de uma porta, devemos usar alguns comandos específicos.

PORTAS DIGITAIS

digitalRead(**pino**) Este comando é responsável pela leitura de dados nas portas digitais disponíveis. Como falamos de leitura de dados digitais, ela será feita como **HIGH** ou **LOW**.

digitalWrite(**pino**, **VALOR**) Se você configurar em void **setup**() um pino como **OUTPUT** (saída de dados), você poderá definir se ela será alta (**HIGH**) ou baixa (**LOW**), assim, você poderá comandar qualquer aplicação que se encontrará conectado em tal **pino**.

PORTAS ANALÓGICAS

Se você for ler dados em uma entrada analógica, você obterá como retorno um valor entre 0 e 1023, pois, o Arduino estudado possui um conversor analógico digital de 10 bits, então, $\mathbf{2^{10} = 1024}$. 1023 é o valor máximo, assim, será colocado na entrada

do pino o valor de tensão máximo de sua tensão de referência (**AREF** – *Analog Reference*).

Mudar a tensão de referência pode ser uma boa ideia, porém, alterando-a em hardware, deve-se também alterar em software. A tensão de referência padrão do Arduino é de 5 V, mas, também há uma tensão de referência interna no valor de 1,1V (no ATMega168), e uma de 2,56V (disponível em apenas algumas plataformas). Pode-se também ser usada uma tensão de referência externa, definida pelo técnico.

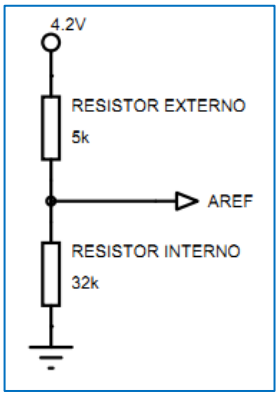
Se por exemplo, for usada a tensão padrão (5V), dizemos que:

5V ÷ 1024 = 0,0048V ≈ 5mV

Isso significa que a cada bit será adicionado 5mV à entrada e que o Arduino apenas conseguirá detectar tensões superiores à 5mV. Alterando o valor de referência é possível obter uma maior sensibilidade, no caso de sensores que não necessitam uma tensão muito alta e uma grande precisão.

OBSERVAÇÃO!

O recomendado é caso for usada uma tensão de referência externa, que seja colocado um resistor de 5kOms, assim, caso possua alguma configuração errada no software e o microcontrolador não conseguir entender, isso previne que ocorra algum dano no equipamento, porém, internamente, há uma resistência no valor de 32kOms, isso significa que você terá um divisor de tensão na entrada do equipamento, ou seja, a tensão de referência será diferente da tensão aplicada antes do resistor de 5kOms.



Exemplo: Se for aplicada uma tensão de 4.2V na entrada do equipamento, a tensão de referência será:

$$AREF = \frac{4,2V \; x \; 32kOms}{32kOms + 5kOms} = 3,63V$$

Se for definido o uso de uma tensão de referência externa, as portas disponíveis com valores de 5V e 3,3V serão fechadas e passarão a não funcionais mais.

analogReference(**TIPO**) Em software, este comando é usado para definir qual é a tensão de referência que será usado em hardware. Ou seja, se o **TIPO** será padrão (**DEFAULT**), interna (**INTERNAL**) ou externa (**EXTERNAL**).

analogRead**(pino)** Este comando é usado para realizar a leitura de alguma entrada analógica. Uma entrada analógica não precisa ser definida na função de inicialização como de entrada ou saída.

O uso desta função retorna um valor inteiro, que como já foi informado, será de 0 a 1023 unidades, sendo equivalente um valor aproximado de 5mV por unidade. Para realizar a leitura de uma entrada digital é necessário um tempo aproximado de 100us.

analogWrite(**pino**, **valor**) Possibilita usar os pinos PWM (Pulse Width Modulation) do Arduino. Esta funcionalidade serve tanto para variar o valor de um LED quanto para variar a velocidade de um motor. O sinal PWM se mantém até ser modificado por outra instrução. A frequência de um sinal PWM é aproximadamente 490Hz. Os **pinos** PWM no Arduino Uno são 3, 5, 6, 9, 10 e 11. O **valor** pode alterar entre **0** (sempre desligado) e **255** (apresenta um sinal constante). Haverá mais explicações sobre sinais PWM posteriormente.

TEMPO

delay(**tempo em ms**) Este comando possibilita uma pausa do programa em execução em uma quantidade de milissegundos específica que é nomeada no campo **tempo em ms**.

delayMicroseconds(**tempo em us**) Este comando possibilita uma pausa do programa em execução em uma quantidade de microssegundos específica que é nomeada no campo **tempo em us**.

millis() Este comando possibilita o retorno da quantidade de tempo que passou, em milissegundos, desde que o programa atual começou a ser executado. Para usar este comando, é necessário o uso da variável ***unsigned long***.

*Observe o **EXEMPLO 2** para podermos prosseguir com os ensinamentos:*

```
/*
Entrada Analógica
Demonstra a leitura de um sensor analógico (A0) e a
emissão de luz de um LED (13).
O tempo que o LED permanecerá acesso dependerá do valor
obtido do sensor.
Este exemplo de código é de domínio público.
Modificado por Micael Bronzatti Gaier
*/
// #include <SoftwareSerial.h>
#define sensor A0 // seleciona o pino de entrada do
//potenciômetro e o identifica como sensor
int led = 13;      // seleciona o pino para o LED
int valSensor;  // variável que armazena o valor do sensor

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

```
void loop() {
valSensor = analogRead(sensor); // lê sensor e define
// valor como variável global
Serial.println(valSensor); // envia valor do sensor //
para porta serial
piscarLED(); // função secundária
 }
void piscarLED(){
digitalWrite(led, HIGH); // ligar o LED
delay(valSensor);  //pausa de acordo com o valor da
//variável
digitalWrite(led, LOW); // apagar o LED
delay(valSensor);
    }
```

BIBLIOTECAS

Você, como programador, deve saber que devemos simplificar ao máximo nossos códigos visando economizar espaço na memória do programa. Para isso quando você obtém um equipamento, sensor, etc., o fabricante muitas vezes fornece uma biblioteca para facilitar a programação, ou seja, invés de você realizar uma longa e extensa programação, você utiliza a biblioteca e simplifica o código.

Quando você obtém uma biblioteca e a armazena na pasta de bibliotecas do Arduino, ao abrir um programa e clicar no menu em **Sketch** > **Import Library**, você poderá escolher a biblioteca que desejar. Quando escolher, aparecerá no seu programa algo como no exemplo acima, porém, sem ser comentado.

```
#include <NomeDaBiblioteca.h>
```

Quando adicionar uma biblioteca ao seu programa, é necessário verificar as instruções corretas para realizar a programação.

#include e #define

Quando você usa **#include**, você está dizendo ao programa que ele deve incluir o arquivo-cabeçalho ao programa.

O uso da ferramenta **#define** é interessante para ser usado com o Arduino. Observando o **exemplo 2**, você percebe que logo após a declaração da ferramenta, é colocado um nome em geral. Após ele foi usado o local onde está o sensor no Arduino, o pino **A0**. Com isso, você percebe que em todo o programa quando quiser se referir ao pino **A0,** se usa a palavra **sensor**. Assim, podemos definir:

#define *nomeVariavel* dados *(OBS: não é usado '**;**' no final)*

Com isso dizemos que podemos usar um nome qualquer, invés de colocar os dados diretamente no programa. Isso facilita a programação, pois ela ficará mais legível e caso seja necessário alterar o valor dos dados, será necessário modificar apenas uma vez. A ferramenta **#define**, é uma **ferramenta local**, ou seja, você a usa apenas para programar, quando você for compilar o programa, o nome da variável será substituído automaticamente pelo dado contido, assim, você acaba economizando espaço na memória do microcontrolador.

VARIÁVEIS E MODIFICADORES

VARIÁVEIS

Os nomes das variáveis apenas devem obedecer algumas regras: elas podem começar com letras ou sublinhado (_) e não podem ter nome idêntico a alguma das palavras reservadas pelo programa ou de alguma biblioteca.

As variáveis devem ser declaradas antes de serem usadas. Observe:

tipo_de_variável *lista_de_variáveis*

Ex: **int** *ledPin, potenciômetro*

CLASSE DE VARIÁVEIS

Um código que se encontra dentro de outra função, entre { e }, se encontra isolado de todo o resto do código e os dados contidos lá apenas podem ser armazenados através de variáveis. Existem três classes de variáveis: locais, globais e estáticas.

VARIÁVEIS LOCAIS

Quando uma variável é declarada dentro de uma função específica, ela é denominada variável local. Estas variáveis apenas existem enquanto o bloco onde está armazenada estiver sendo executado. A partir do momento em que o programa voltar à função principal, esta variável deixará de existir.

VARIÁVEIS GLOBAIS

Uma variável global é aquela variável que é conhecida por todo o programa, ou seja, independente da função que estiver sendo executada ela será reconhecida e rodará normalmente. Uma variável global ela é declarada antes mesmo da função void **setup**().

VARIÁVEIS ESTÁTICAS

Funcionam de forma parecida com as variáveis globais conservando o valor durante a execução de outras funções, porém, só são reconhecidas dentro da função onde é declarada. Como exemplo, podemos dizer que uma variável estática é uma variável declarada dentro da lista de parâmetros de uma função.

TIPOS DE DADOS E MODIFICADORES

Em linguagem C, e também em Arduino, existe 5 tipos de dados: **char, int, float, void, double**.

void é usado apenas na declaração de funções, indica que a função chamada não retornará nenhum valor para a função que a chamou.

*Exemplo: void **loop**()*

boolean – *variáveis booleanas* estas variáveis memorizam um de dois valores: verdadeiro (true) ou falso (false).

Exemplo: condicao = false;

char é um tipo de dado que dedica 1 byte de memória para armazenar o valor de um caractere.

Exemplo: palavra = 'Arduino';

byte byte memoriza um número de 8-bits, de 0 a 255. Esse tipo de dados não memoria número negativos.

Exemplo: byte b = B10010 ("B" significa que está em formato binário)

int é a variável padrão do programa. Esta variável consegue memorizar dados de -32768 até 32767. Esta variável consegue memorizar números negativos através da propriedade matemática chamada **complemento de dois.**

Exemplo: int ledPin = 13; int nomeVariavel = valorVariavel;

string é um vetor de caracteres terminado em um caractere nulo. Você pode usar uma string para memorizar dados obtidos em forma de caracteres. Numa string você denomina o nome e a quantidade máxima de caracteres que você pretende usar mais um caractere reservado para o caractere nulo. Usando uma string você pode modificar um caractere em particular no programa.

Exemplo: char str1[8] = { 'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0' };
char str2[15] = "arduino";

TIPO	TAMANHO (BITS)	INTERVALO
char	8	-128 a 127
int	16	-32768 a 32767
float	32	3,4E-38 a 3,4E+38
double	64	1,7E-308 a 1,7E+308
void	0	sem valor

MODIFICADORES

Para cada tipo de variável, existem os modificadores de tipos, sendo eles: **signed, unsigned, short, long**. Porém, em **float** não se pode aplicar nenhum modificador, e no **double** o único aceito é o **long**.

Os quatro modificadores podem ser aplicados a qualquer inteiro. O objetivo de **short** e **long** é que possam oferecer tamanhos diferentes de inteiros. Na realidade, o **int** terá o tamanho natural de uma determinada máquina, a única restrição é que

shorts e **ints** devem ocupar pelo menos 16 bits, **longs** pelo menos 32 bits, e que **short** não pode ser maior que **int**, que não pode ser maior que **long**.

Observe a tabela.

TIPO	NUMERO DE BITS	INTERVALO	
		INICIO	FIM
char	8	-128	127
unsigned char	8	0	255
signed char	8	-128	127
int	16	-32.768	32.767
unsigned int	16	0	65.535
signed int	16	-32.768	32.767
short int	16	-32.768	32.767
unsigned short int	16	0	65.535
signed short int	16	-32.768	32.767
long int	32	-2.147.483.648	2.147.483.647
signed long int	32	-2.147.483.648	2.147.483.647
unsigned long int	32	0	4.294.967.295
float	32	3,4E-38	3.4E+38
double	64	1,7E-308	1,7E+308
long double	80	3,4E-4932	3,4E+4932

FUNÇÕES I

Anteriormente, já foi explicado um pouco a respeito sobre o void **setup**() e o void **loop**(), que mesmo não utilizando-os, eles precisam ser declarados.

Uma função é uma unidade autônoma do código, que foi feita para cumprir uma atividade em particular. Um programa consiste em várias funções. Uma função se resume em:

```
tipo nome(declaração de parâmetros){
comando1;
comando2;
...
}
```

Existem dois tipos de funções: as que retornam algum valor para a função onde está inserida e as funções que não retornam nenhum valor.

Tipo int Uma função **int** retorna um valor para a função que a chamou. Há duas formas de retornar de uma função, uma delas é quando o finalizador de função (***}***) é encontrado, ou quando for feito o uso da declaração **return**.

A declaração **return** retorna um valor especificado para a função que a chamou.

Exemplo:

```
int checkSensor(){
   if (analogRead(A0) > 400) {      // se a leitura do pino A0 for maior que 400
      return 1;                     // retorna 1 (verdadeiro)
   else{
      return 0; }                   // se não for, retorne 0 (falso)
}
```

Tipo void uma função tipo **void** não retorna nenhum valor para a função que a chamou, ou seja, as ações executadas nesta função não resultaram em números binários, variáveis booleanas ou caracteres para a função principal. Com o uso de **void** não é permitido usar a declaração **return**.

Usar funções pode simplificar o código e o deixar mais organizado. No **exemplo 2** percebemos que foi criada uma função apenas para o LED piscar e que foi inserida na função *void **loop()***, com isso, se o código fosse extenso e seria necessário o LED piscar várias vezes durante o programa, aquela parte do código não precisaria ficar sendo repetida, assim, apenas chamaria a função *piscarLED(),* então quando ela terminasse de ser executada voltaria ao mesmo ponto em que parou.

Como programar em Arduino

Agora que você já sabe identificar **comentários**, **bibliotecas**, **variáveis**, consegue compreender como lidar com o **tempo**, sabe criar **funções,** identificar o **corpo do programa** e **enviar e receber dados de entradas digitais e analógicas**, podemos começar a **programar em Arduino**.

OPERADORES BOOLEANOS, DE COMPARAÇÃO E INCREMENTO E DECREMENTO

Um passo importante para podermos programar corretamente é de fazer o uso adequado dos operadores booleanos e os de comparação.

OPERADORES DE COMPARAÇÃO

x é igual a y:	x **==** y
x é diferente de 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:	x **>=** y

OPERADORES BOOLEANOS

Os operadores booleanos verificam se uma função é verdadeira ou não.

OPERADOR E (&&)

A condição apenas será verdadeira se todos os itens foram verdadeiros.
Exemplo: if (digitalRead(2) == 1 && digitalRead(3) == 1){
... }
*Se a leitura digital dos pinos 2 e 3 forem **1** (**HIGH**), a condição é verdadeira.*

OPERADOR OU (||)

A condição será verdadeira se algum dos itens for verdadeiro.
Exemplo: if (x>0 || y>0){
... }
*Se **x** ou **y** for maior que 0, a condição será verdadeira.*

OPERADOR NÃO (!)

A condição será verdadeira se o operador for falso.
Exemplo: if (!x){
... }
*Se **x** for falso, a condição é verdadeira.*

OPERADORES DE INCREMENTO E DECREMENTO

Em Linguagem C trabalhamos com operadores **unários e binários**. Operadores unários agem em apenas uma variável e a modifica ou não, enquanto os operadores binários utilizam duas variáveis, pegam seus valores sem alterá-los e retorna um terceiro valor. A soma é um exemplo de operador binário. Os operadores de **incremento e decremento são operadores unários**, pois **alteram a variável em que estão aplicados.** Então dizemos:

x++ **x--**

Que isso é equivalente a:

x++ > **x=x+1**

x-- > **x=x-1**

Estes operadores podem ser **pré-fixados** ou **pós-fixados**. Os *pré-fixados eles incrementam e retornam o valor da variável já incrementada*, enquanto os *pós-fixados eles retornam o valor da variável sem o incremento e depois incrementam a variável.*

Pré-fixado	*Pós-fixado*
Se: x=48 y=++x	**Se:** x=48 y=x++
Então: x=49 y=49	**Então:** x=49 y=48

Os operadores de incremento e decremento são de grande utilidade neste tipo de linguagem quando se trata de realização de testes, como veremos adiante.

OUTROS USOS

x += y > ***x = x + y***

x – = y > ***x = x – y***

x *= y > ***x = x * y***

x /= y > ***x = x / y***

ESTRUTURAS DE CONTROLE DE FLUXO

Podemos dizer que as estruturas de controle de fluxo são a parte mais importante da programação em Arduino, pois toda a programação é executada em torno delas.*

IF

IF testa se certa condição foi alcançada, como por exemplo, se uma entrada analógica obteve um número específico. Se o resultado da condição for 0 (falsa), ela não será executada, caso o resultado seja 1 (verdadeira), a função será executada. O formato desta estrutura de controle de fluxo é:

```
if (certaCondicao) {
        // comandos... }
```

IF... ELSE

Usar **if/else** permite um controle maior sobre a estrutura, sendo que poderá ser realizado vários testes sem sair de uma única função. Podemos pensar no **else** como sendo um complemento ao comando **if.**

```
if (certaCondicao) {
        // comando A… }
else {
        // comando B...  }
```

Podemos interpretar a estrutura da seguinte maneira: "***se*** *a* ***certaCondicao*** *for verdadeira (1) execute o* ***comando A****,* ***senão*** *(se for falsa, 0), execute o* ***comando B***".

OBSERVAÇÃO!

Quando você estiver trabalhando com várias condições, onde não se restringe a apenas duas hipóteses, dentro da estrutura de controle ***else****, você pode colocar mais funções* ***if/else*** *e assim por diante. Veja:*

```
if (Condicao1) {
  // comando A… }
else if (Condicao2) {
  // comando B...  }
        else {
        // comando C }
```

FOR

O estrutura for é uma das estruturas que se trabalha com loops de repetição. Esta estrutura normalmente é usada para repetir um bloco de informações definidas na função. Enquanto a condição for verdadeira, as informações contidas na função serão executadas. Sua forma geral é:

```
for (inicialização; condição; incremento)
{
  //instrução(ou instruções);  }
```

Vamos entender os blocos desta estrutura:

INICIALIZAÇÃO: a inicialização é o que ocorre primeiro e apenas uma vez.

CONDIÇÃO: o bloco for trabalha com loops de repetição, cada vez que a função for repetida, a condição será testada, se ela for verdadeira (1), executará o que se encontra dentro das chaves.

INCREMENTO: se a condição for verdadeira, o incremento será executado, caso contrário, se a condição for falsa, o loop é terminado.

Caso ainda não conseguiu entender a estrutura for equivale a mesma coisa que o seguinte código:

```
inicialização;          // primeiro: realiza-se a inicialização
```

```
if (condição) {          // segundo: verifica se a condição é verdadeira
      declaração;    // terceiro: executa a declaração contida na função
      incremento;    // realiza o incremento
      "Volta para o comando if"
      }
```

EXEMPLO 3:

```
// Dimmer de um LED usando um pino PWM como saída

int pinoPWM = 10; // LED ligado em série com um resistor de 1k
                  //ao pino 10
void setup()
{
   // não é necessário setup
}
void loop()
{
   for (int i= 0; i <= 255; i++){
   // a variável i é incrementada até que irá valer 255,
   //depois o loop é terminado
      analogWrite(pinoPWM, i);
   //o valor de i é escrito no pinoPWM
      delay(10);
   }
}
```

SWITCH CASE

O Switch Cace permite programar diferentes blocos de instruções para diferentes condições. Por exemplo, dizemos que você possui um sensor analógico que emite diferentes valores e cada valor indica que você necessita tomar uma ação específica, ou seja, uma instrução diferente. Sua forma geral é:

```
switch (valor) {
  case 1:
    //fazer algo quando valor é igual a 1
  break;

  case 2:
    //fazer algo quando valor é igual a 2
  break;

  default:
    // se nenhum caso se encaixa, fazer algo como padrão
}
```

Quando uma condição for verdadeira, ***break*** é responsável por interromper o código e seguir a programação adiante. Usa-se ***default*** caso nenhuma das condições especificadas forem verdadeiras, e então o bloco contido será executado.

EXEMPLO 4:

```
/* Leve em consideração que você possui um equipamento que
possui em sensor analógico que emite valores diferentes para
cada situação. Assim, cada um desses sensores deverá ligar um
LED específico conectado nos pinos 08 a 11.
Código feito por: Micael B. Gaier
*/

#define sensor A0 //sensor
// LEDs conectados do pino 08 ao 11

void setup(){
  for (int pino = 8; pino < 12; pino++) {
    pinMode(pino, OUTPUT); // inicialização dos LEDs
  }
}

void loop(){
  // leitura do sensor analógico
  int valor = analogRead(sensor);
   switch(valor) {
     case 205 : // condição 1
     digitalWrite(8,HIGH);
     break;

     case 409 : // condição 2
     digitalWrite(9, HIGH);
     break;

     case 615 : // condição 3
     digitalWrite(10, HIGH);
     break;

     case 830 : // condição 4
     digitalWrite(11, HIGH);
     break;

     default : // condição padrão
     // ligar LEDs
     for (int pino = 8; pino < 12; pino++){
       digitalWrite(pino, HIGH);
     }
     // apagar LEDs
     for (int pino = 8; pino < 12; pino++){
       digitalWrite(pino, LOW);
     } } }
```

WHILE

A estrutura WHILE funciona como um loop de repetição. Enquanto a condição contida dentro dos parênteses permanecer verdadeira, a estrutura será executada. A partir do momento que a condição for falsa, o programa seguirá normalmente. Sua forma geral é:

while *(condição) {*
// instrução 1...
// intrução 2...
}

A condição será booleana, ou seja, será avaliada se é 1 (verdadeira) ou 0 (falsa). ***EXEMPLO 5:***

```
/* Controlar três saídas digitais de acordo com o valor obtido
de uma entrada analógica
Feito por Micael B. Gaier */

 #define analogPin A0
 #define digPin01 4
 #define digPin02 5
 #define digPin03 6

 void setup(){
   // inicialização dos pinos digitais
   for (int i = 4; i<7; i++){
     pinMode(i, OUTPUT);
   }
 }

 void loop(){
   int valor = analogRead(analogPin); // leitura pino analógico

   while (valor < 900){ // executar se 'valor' for menor que
900
     if (valor < 200){
       digitalWrite(digPin01, HIGH);
     }
     else if ( valor >= 200 && valor < 600){
       digitalWrite(digPin02, HIGH);
     }
     else {
       digitalWrite(digPin03, HIGH);
     }
   }
   for (int i = 4; i<7; i++){
     digitalWrite(i, LOW);
   }
 }
```

DO – WHILE

A estrutura de controle **do - while** é a ultima estrutura de controle de loop repetitivo que estudaremos. Esta estrutura funciona de forma semelhante às estruturas **for** e **while**, porém, ao contrário. Observe sua forma geral:

```
do {
   // instrução 1...
   // instrução 2...
} while (testa a condição);
```

Esta estrutura primeiro realiza as instruções, e após ela ser executada é verificado se a condição é valida, se ela ainda for válida, ela volta a executar o bloco novamente. ***EXEMPLO 6:***

```
/* Envia um sinal analógico a um equipamento, de acordo com o
valor obtido de uma entrada digital.
Feito por Micael B. Gaier */
#define digitalPin 5
#define analogPin A0
// para enviar o sinal analógico, o sinal digital deverá ser baixo
int valIdeal = LOW;

void setup(){
  pinMode(digitalPin, INPUT);
}

void loop(){
  int val;
  do {
    analogWrite(analogPin, 255);
    val = digitalRead(digitalPin);
  } while (val == valIdeal);
}
```

COMUNICAÇÃO SERIAL

Se você perceber em placas Arduino como Uno, Duemilanove, Nano e Mega, você verá que há um CI responsável por converter o sinal da porta serial de hardware para Universal Serial Bus (USB), para poder ser usado em qualquer dispositivo que possua esta sistema, como seu computador, por exemplo. Na comunicação serial, é possível você obter todos os dados que o Arduino está gerando ou recebendo e enviar ao computador para simplesmente analisar os dados ou rodar algum outro programa.

Quando você conecta um dispositivo à porta serial do Arduino, é necessário uma “taxa de bits por segundo” para a transmissão serial. O Arduino consegue emitir esta taxa nos seguintes valores: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, ou 115200 bits por segundo. Há equipamentos que apenas receberão

estes dados caso o Arduino esteja configurado em uma taxa de bits especifica. Um computador, por exemplo, é configurado por padrão por receber dados da USB numa taxa de 9600.

Da mesma maneira que iniciamos uma entrada digital na função *void* ***setup()***, devemos iniciar o comunicação serial, porém usamos a seguinte função:

Serial.begin*(taxa);*

Veja a seguir alguns comandos relacionados a comunicação serial.

Serial.available*();*

Obtém o número de bytes disponíveis para a porta serial. Este comando retorna o número de bytes disponível para leitura.

Serial.read*();*

Obtêm os dados da porta serial para leitura.

Serial.print*();*

Envia dados para a porta serial como texto legível ASII. Este comando pode possuir diferentes formas. Por exemplo:

- **Serial**.print**(78)** *resulta em* **"78"**
- **Serial**.print**(1.23456)** *resulta em* **"1.23"**
- **Serial**.print**(byte(78))** *resulta* em **"N"** *(em ASCII seu valor é 78)*
- **Serial**.print**('N')** *resulta em* **"N"**
- **Serial**.print**("Ola Mundo.")** *resulta em* **"Ola Mundo."**

OBSERVAÇÃO!
Para nos facilitar a tarefa de programar, existem várias constantes. São caracteres que podem ser usados como qualquer outro. Observe:

Caractere	***Significado***
\n	*Nova Linha*
\t	*Tabulação Horizontal ("tab")*
\a	*Tabulação Vertical*
BIN	*Binário ou Base 2*
OCT	*Base 8*
HEX	*Hexadecimal ou Base 16*
DEC	*Decimal, ou Base 10*
BYTE	*Resulta em Byte*

Serial.println*();*

Envia dados para a porta serial, porém, com um caractere de retorno. Este comando funciona da mesma forma que **Serial**.print*().*

Exemplo 7:

```
/* Função de comunicação serial.*/
#define pot A0
int var;

void setup(){
  // iniciarndo porta serial
  Serial.begin(9600);
}

void loop(){
  // obtendo valor do potenciômetro
  var = analogRead(pot);

  if (Serial.available() > 0){
  // imprimindo frase na porta serial
  Serial.println("O valor obtido do potenciômetro e: ");
  // imprimindo valor da variável
  Serial.println(var);
  }

  delay(500);
  // lendo valores obtidos da porta serial
  int valSerial = Serial.read();
 // imprimindo texto na porta serial
  Serial.print("Foi obtido da porta serial o seguinte valor: ")
 // imprimindo valores obtidos da porta
  Serial.println(valSerial, DEC);
}
```

OPERADORES MATEMÁTICOS E TRIGONOMETRIA

No Arduino também podemos realizar operações matemáticas e podemos trabalhar com trigonometria. Veja.

OPERADORES MATEMÁTICOS

min*(x, y)* Calcula o mínimo de dois números, sendo *x* o primeiro número e *y* o segundo número.

Ex: min(val, 100);

max*(x,y)* Calcula o máximo de dois números, sendo *x* o primeiro número e *y* o segundo.

Ex: max(1, 300); *(irá retornar 301)*

abs*(x)* Retorna o valor absoluto de uma função. Ou seja se *x* for maior que zero, retornará *x*, se *x* for menor que 0, retornará o mesmo valor, porém positivo.

Ex: abs(-5); *(a função retornará '5')*

constrain*(x,a,b)* Limita um número dentro de uma faixa de valores.

x = é o número a limitar, o valor de referência;
a = é o valor que será retornado caso x seja menor que a;
b = é o valor que será retornado caso x seja maior que b;
Ex: constrain(sens,1,3);

map*(valor, deBaixa, deAlta, paraBaixa, paraAlta)* Mapeia um valor de um intervalo para outro. Este operador matemático pode ser facilmente usado quando você desejar que os valores obtidos de uma entrada analógica (de 0 a 1023) controlem um valor de saída de uma entrada digital (de 0 a 255), ou seja, você *"transforma"* um número em uma faixa de valores para outra faixa diferente.

valor = valor a ser mapeado;
deBaixa = o limite inferior atual da faixa atual do valor;
deAlta = o limite superior da faixa atual do valor;
paraBaixa = o limite inferior do valor de destino;
paraAlta = o limite superior do valor de destino;

Ex:

```
void setup() {}

void loop() {
  int val = analogRead(0);
  val = map(val, 0, 1023, 0, 255);
  analogWrite(9, val);
}
```

PARA ENTENDER!

Dizemos que você possui um sensor analógico (0 a 1023) que está controlando a velocidade de giro de um motor conectado numa entrada PWM (0 a 255). Quando o sensor analógico emitir o valor de 0, o resultado na saída PWM será 0, enquanto se o sensor analógico emitir 1023, o resultado em PWM será 255. Caso seja obtido um valor intermediário do sensor, a saída PWM também obterá um valor intermediário equivalente, mas em sua faixa de valores.

OBSERVAÇÃO: Esta função não limita a faixa de valores, se você estiver lidando com *variáveis* em **valor**, e o seu limite superior de **deAlta** é 1023 e o **paraAlta** é 255, caso você inserir um valor superior a 1023, o valor da faixa e saída também será maior que 255. Por isso pode ser usada a função constrain*(x,a,b)* para auxiliar e limitar esta faixa de valores.

pow*(base, expoente)* Calcula o valor de um número elevado a uma potência.

base = um número qualquer (float)
expoente = a potência que a base será elevada (float)
RETORNO DA FUNÇÃO: double

Ex:
pow*(7,2) ;* - O resultado obtido será um double no valor 49

.

sqrt*(x)* Permite o calculo da raiz quadrada de um determinado valor.
Ex: valor = **sqrt*(49)* ;** - *valor será 7, pois* $\sqrt{49} = 7$.

TRIGONOMETRIA

sin*(rad)* Calcula o seno de um ângulo (em radianos).
rad = valor do ângulo em radianos (float)
RETORNO DA FUNÇÂO: a função retorna um valor entre -1 e 1 (double).

cos*(rad)* Calcula o cosseno de um ângulo (em radianos).
rad = valor do ângulo em radianos (float)
RETORNO DA FUNÇÂO: a função retorna um valor entre -1 e 1 (double).

tan*(rad)* Calcula a tangente de um ângulo (em radianos).
rad = valor do ângulo em radianos (float)
RETORNO DA FUNÇÂO: a função retorna um valor infinito negativamente ou positivamente (double).

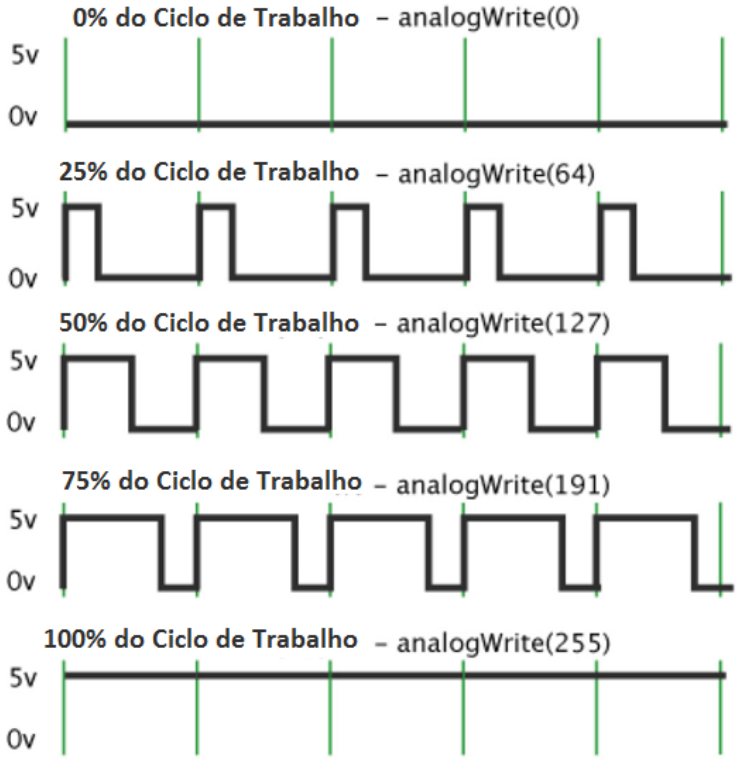
OBSERVAÇÃO! As funções **Serial**.print() e **Serial**.println() não suportam a impressão de floats.

ENTRADA E SAIDA DE DADOS II

SINAIS PWM

Pulse Width Modulation, Largura de Modulação de Pulso, ou simplesmente PWM é a técnica de obtém resultados analógicos através de meios digitais. O controle digital é usado criando uma onda quadrada, um sinal oscilando entre “ligado” ou “desligado”, “0” ou “1”. Por uma porção de tempo o sinal permanece “alto” enquanto pelo resto deste tempo ele permanece “baixo”. A duração desde “tempo” é chamada modulação de pulso.

Em um Arduino, a frequência de um sinal PWM é cerca de 490Hz, no caso, a modulação de pulso oscilaria a cada 2 milissegundos. Se você aplicar um sinal através de ***analogWrite()*** com valores entre 0 e 255, você obteria os valores da tabela a seguir em seu ciclo de trabalho. Observe o Exemplo 8.



EXEMPLO 8

```
/* Controla o brilho de um LED através de um sinal PWM.
 Este exemplo de código é de domínio público e está contido dentro do software
do Arduino.
*/

int ledPin = 9; // LED no pino PWM 9

void setup()  {
  // não é necessário realizar declarações
}
void loop()  {
  // aumenta o brilho do LED de 0 a 255, num incremento de valor +5
  for(int fadeValue = 0 ; fadeValue <= 255; fadeValue +=5) {
    analogWrite(ledPin, fadeValue);
    // aguarda 30 milissegundos para o efeito de fading
    delay(30);
  }
  // diminui o brilho do LED de 0 a 255, num incremento de valor -5
  for(int fadeValue = 255 ; fadeValue >= 0; fadeValue -=5) {
    analogWrite(ledPin, fadeValue);
    delay(30);
  } }
```

pulseIn*(pino,* valor, tempo de resposta*)*

Lê o pulso de um pino.

Por exemplo, se valor é **HIGH**, a função aguardará um tempo específico em *microssegundos* de acordo com o valor do **tempo de resposta**, e quando obtiver o valor específico no pino ele começará uma contagem de tempo e apenas interromperá quando o valor será **LOW**. **A função retornará o comprimento do pulso em microssegundos.**

PARÂMETROS:

pino = número do pino no qual você deseja ler o pulso (int);

valor = HIGH ou LOW (int);

tempo de resposta *(opcional)* = o número de microssegundos a esperar após o pulso iniciar (por padrão é um segundo).

RETORNO DA FUNÇÂO:

O comprimento do pulso em microssegundos ou 0 se o pulso começou sem um tempo de resposta.

EXEMPLO 9

Este exemplo consiste no uso do sensor ultrassônico LV-Max Sonar EZ0 que é capaz de informar se há obstáculos a sua frente até 6,45 metros. Para o sistema funcionar corretamente é necessário um resistor de 100 ohms entre o pino de 5V do Arduino e a Entrada Vin do sensor, e um capacitor eletrolítico de 100µF entre as alimentações (Vin e Gnd).

```
/* Uso do sensor LV-MaxSonar EZ0 para capturar obstáculos a uma distancia de até 6,45m
*/
#define sensorUltra 9
long pwmOnda, pol, cm;
void setup() {
  Serial.begin(9600);
  pinMode(sensorUltra, INPUT);
}

void loop() {
  pwmOnda = pulseIn(sensorUltra, HIGH);
  pol = pwmOnda / 147;
  cm = pol * 2.54;
  Serial.print(pol);
  Serial.print(" polegadas    ");
  Serial.print(cm);
  Serial.println(" cm");
  }
```

Usando o Arduino com Dispositivos

O USO DE ARDUINO EM PROJETOS

O uso do Arduino com dispositivos eletrônicos vem se mostrando muito eficaz. A cada dia o número de dispositivos criados para funcionar exclusivamente com Arduino vem aumentando cada vez mais. O sucesso do uso deste equipamento é devido ao fato de apenas conectar o dispositivo ao Arduino e de realizar uma programação simples em sua interface.

Quando se deseja colocar o estudo de um projeto com dispositivos eletrônicos em prática é necessário realizar uma série de passos fundamentais. Antes de tudo você precisa simular o projeto via software e verificar se ele funcionará adequadamente, após é necessário criar a placa de circuito impresso no computador e então dar inicio ao processo de fabricação da mesma, então é necessário fixar todos os componentes na placa, testar novamente para verificar possíveis erros e após tudo isso que você poderá estar realizando estudos mais aprofundados.

Porém, usando o Arduino todos estes passos anteriores são resumidos e você acaba tendo um melhor desempenho. Isso faz com que muito pesquisadores acabam ganhando tempo em seus estudos e acabam obtendo melhores resultados. Por este motivo a cada dia aumenta o número de Universidades de aderem ao uso do Arduino em seus métodos de pesquisa e ensino.

Pelo Arduino já ter se tornado uma linguagem universal de programação, para qualquer aplicação que você deseja realizar, você encontrará informações úteis em fóruns específicos na internet e poderá encontrar até Shields específicos para cada um dos diversos usos.

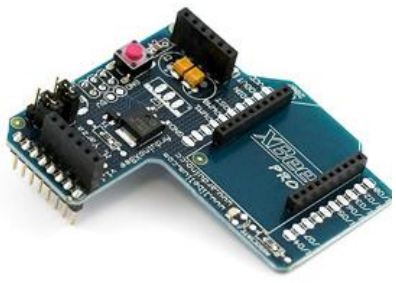
SHIELDS PARA ARDUINO

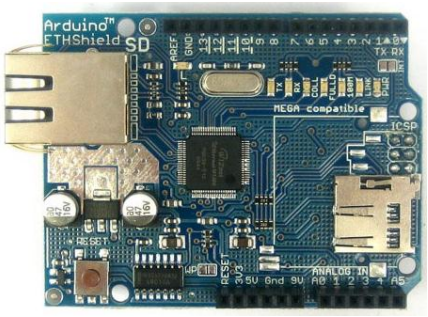
Shields são placas que otimizam as capacidades do Arduino e que podem ser plugadas na parte superior da placa PCB. Os Shields se baseiam no mesmo objetivo do Arduino: fácil de montar, barato de produzir.

Atualmente você pode obter shields de diferentes maneiras, para diferentes usos e de fabricantes diferentes. No final desta apostila, você encontrará uma relação dos principais sites e fabricantes destes produtos.

Exemplos de Shields:

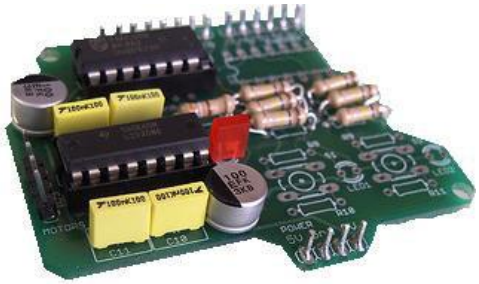
Arduino XBee Shield: esta placa se encaixa diretamente sobre o Arduino Duemilanove e permite comunicação sem fios sobre um protocolo modificado ZigBee da Maxstream.

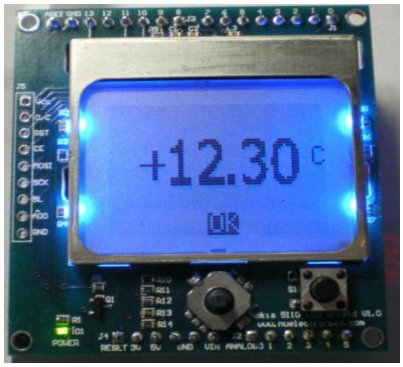




Arduino Ethernet Shield: permite que o Arduino seja conectado à internet, versões mais recentes deste shield também possuem slot para cartão miro-SD, para que possa ser armazenados dados.

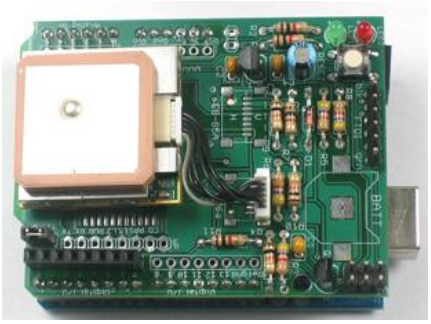
Motor Shield: são shields onde você pode conectar diretamente no Arduino diversos tipos de motores, de diferentes capacidades.





LCD Shield: este shield conecta o Arduino à uma tela LCD permitindo uma maior interação com o usuário.

Bluetooth Shield: são shields que permitem realizar conexão bluetooth com outros dispositivos eletrônicos, como celulares, por exemplo.



GPS Shield: permite a conexão do Arduino com um receptor GPS.

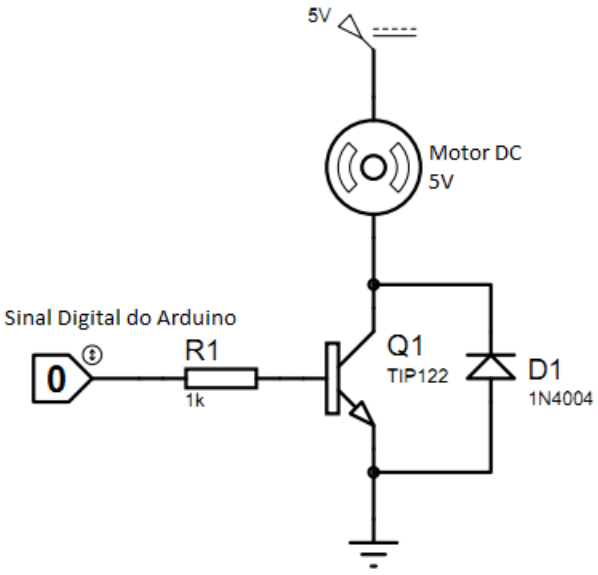
Encontramos hoje no mercado uma vasta lista de shields para Arduino, além destes, podemos encontrar shields que permitem realizar diversas funções como: MP3 Player, conexão com redes wi-fi, shields que simulam funções de celulares, etc.

Normalmente os fabricantes dos shields oferecem bibliotecas de conteúdo para facilitar ainda a programação, assim, muitas vezes não é necessário fazer um estudo muito detalhado para aprender a executar funções específicas.

FAZENDO UM MOTOR DC FUNCIONAR

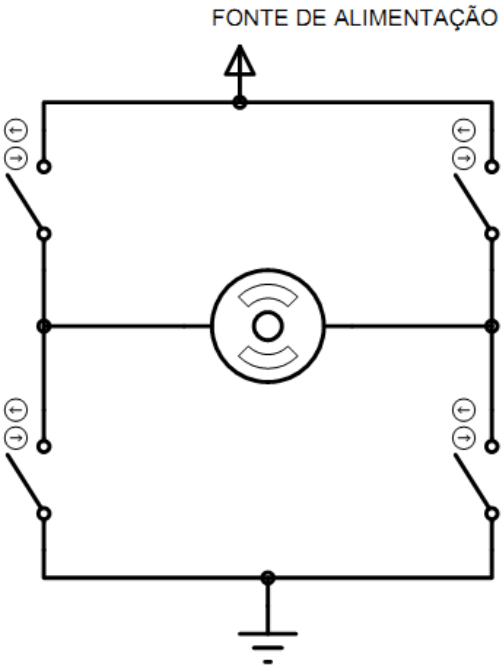
Quando pensamos em controlar um robô através de um sistema inteligente logo pensamos em sua capacidade de locomoção, ou seja, em fazer o uso de motores para movimentá-lo de acordo com sinais enviados pelo microcontrolador. Logo, devemos pensar que este motor não poderá ficar conectado diretamente à fonte de tensão e que devemos criar um “sistema de chaves inteligentes” para comandar o motor de acordo com o sinal enviado do Arduino.

Um Arduino sozinho não é capaz e nem é seguro fazer um motor DC funcionar (o Arduino não é capaz, pois fornece em suas saídas digitais corrente de cerca de 40mA), por isso, devemos obter um driver capaz de fornecer corrente suficiente para o motor DC funcionar adequadamente. No caso de pequenos motores o uso de transistores e diodos pode ser suficiente para fazer um “sistema de chaves inteligentes”. Observe o seguinte esquema.

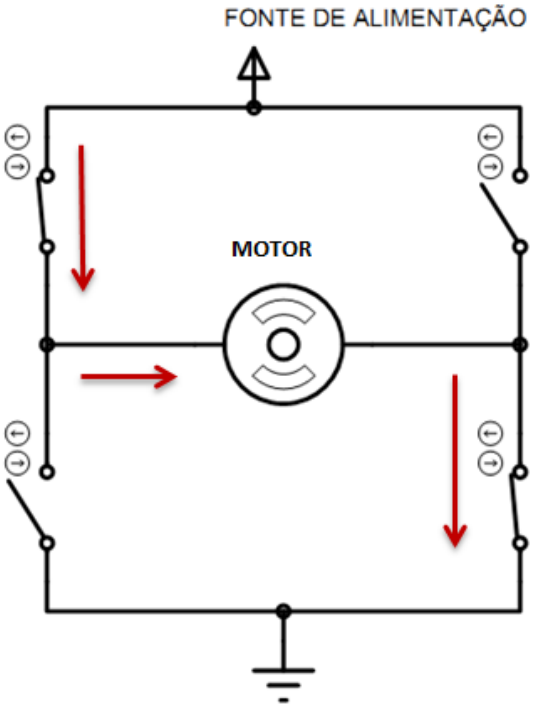


Porém se analisarmos o circuito anterior, observaremos que o motor irá operar sem controle de velocidade e em apenas um sentido, não tornando o sistema totalmente inteligente e eficiente.

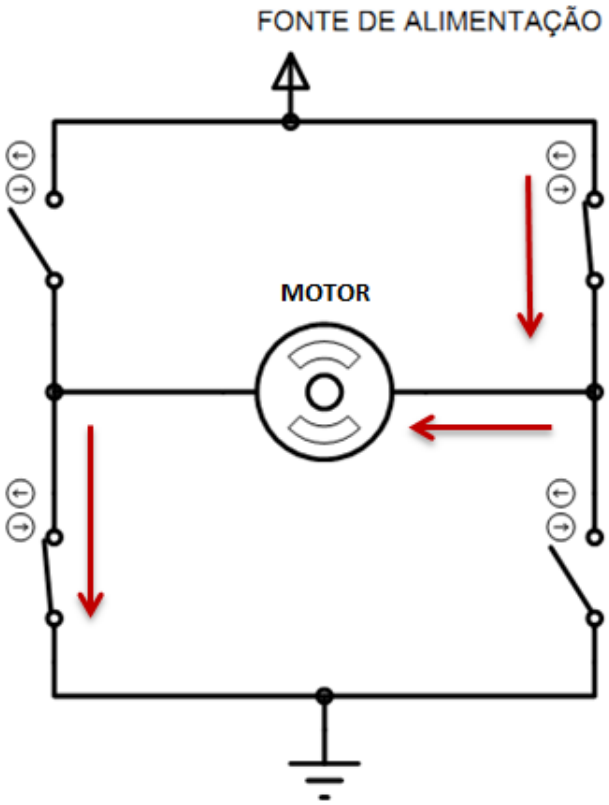
Um motor tem capacidade de movimentação tanto no sentido horário como no sentido anti-horário, e para aproveitarmos isso, devemos apenas criar um jogo de chaves onde podemos inverter o sentido da corrente do circuito.



Se a corrente estiver sendo conduzida de um lado ela deverá seguir o seguinte esquema, fazendo o motor girar em um sentido.



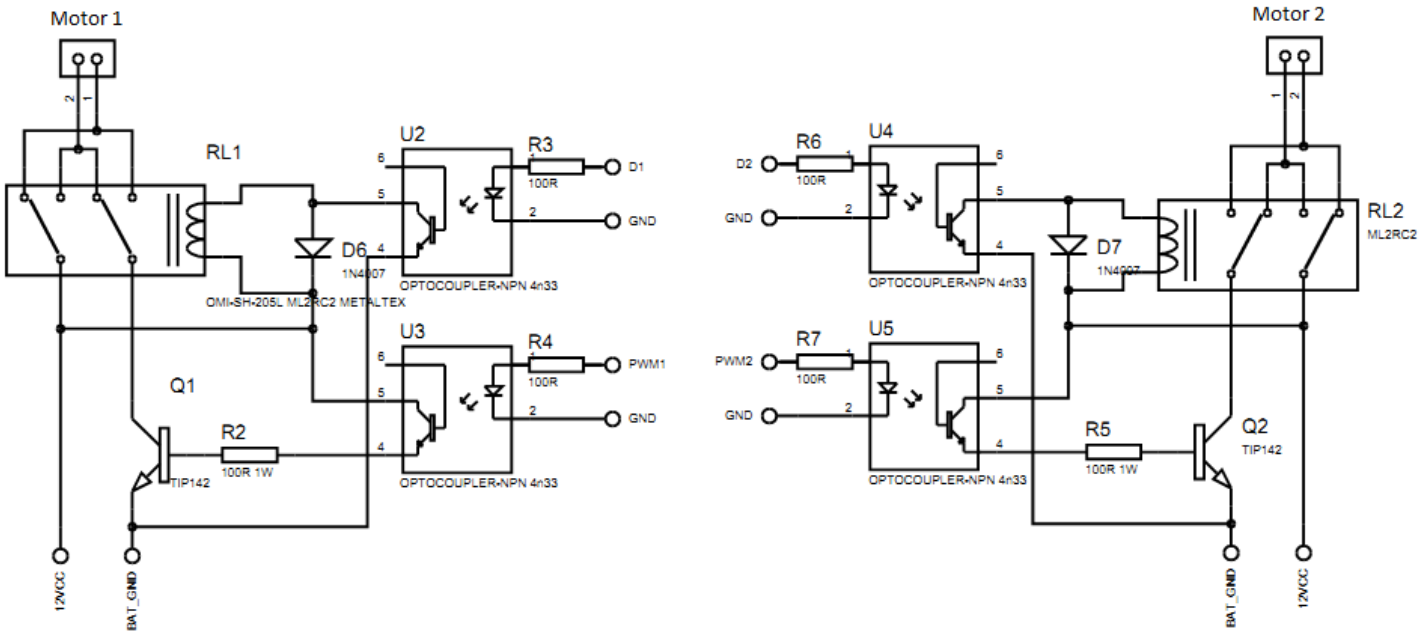
Se a corrente estiver sendo conduzida pelo outro lado ela deverá seguir este esquema, fazendo o motor girar no outro sentido.



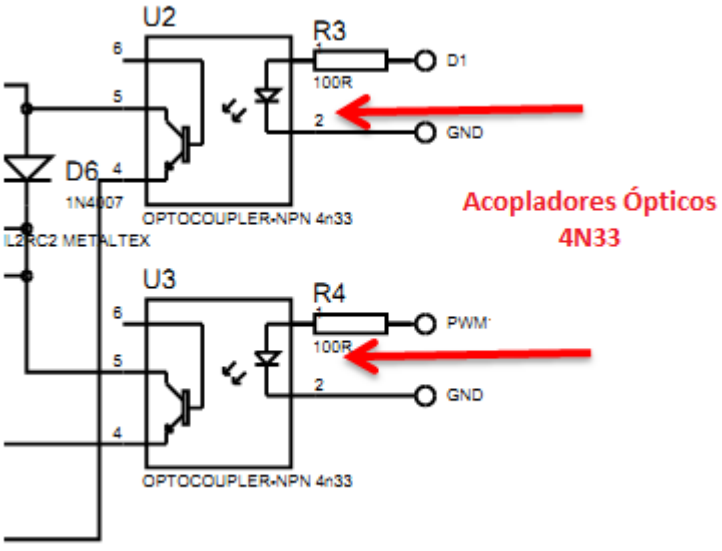
Para um motor ter a habilidade de movimentação nos dois sentidos do seu eixo e também para obter um controle sobre sua velocidade, torque e outros dados técnicos, devemos montar um circuito que seja inteligente para poder realizar estes fundamentos básicos apresentados através de sinais enviados do Arduino. Se observarmos atentamente o esquema de chaves, veremos que ele se torna similar com um “H”, assim, dando a estes estudos o nome de **Ponte H** (*H-Bridge, em inglês*).

Há várias maneiras de se montar uma Ponte H. Ela pode ser feita tanto com relés, como com transistores e até com CI’s (circuitos inteligentes) já criados especialmente para isso. Não há como apresentar um modelo de Ponte H e dizer que ela funcionará adequadamente com qualquer circuito que for, pois, ao montar o circuito, devemos levar em consideração os dados técnicos do motor que está sendo usado, as especificações do Arduino, dos dispositivos que serão utilizados e do ambiente em que o sistema atuará.

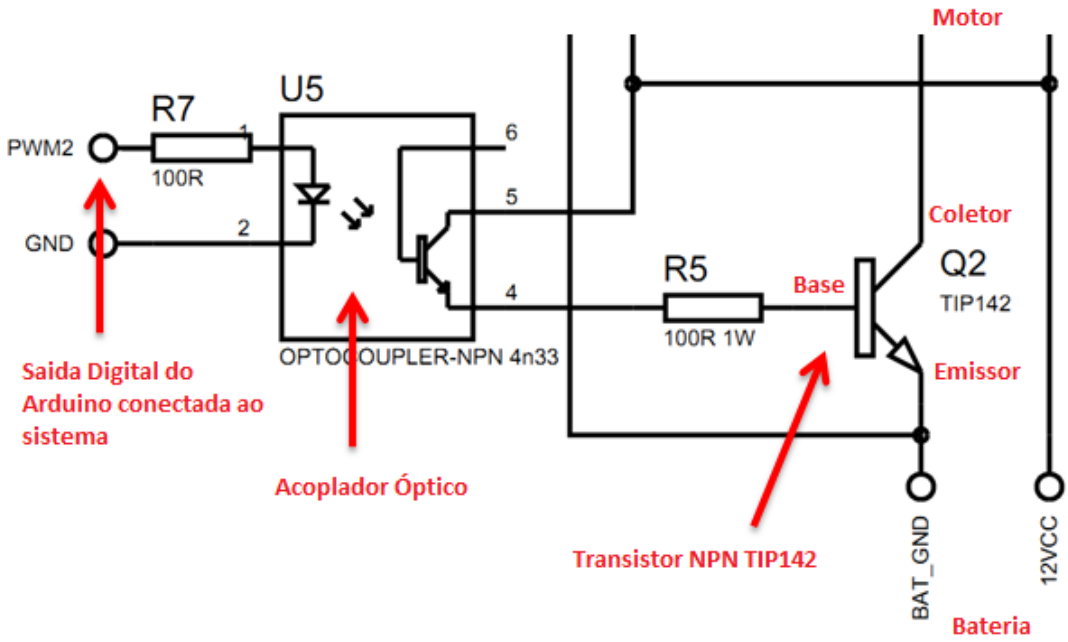
Como exemplo nós usaremos uma Ponte H dupla desenvolvida no IFMT para comandar dois motores DC em um Robô Sumo. Observe o esquema.



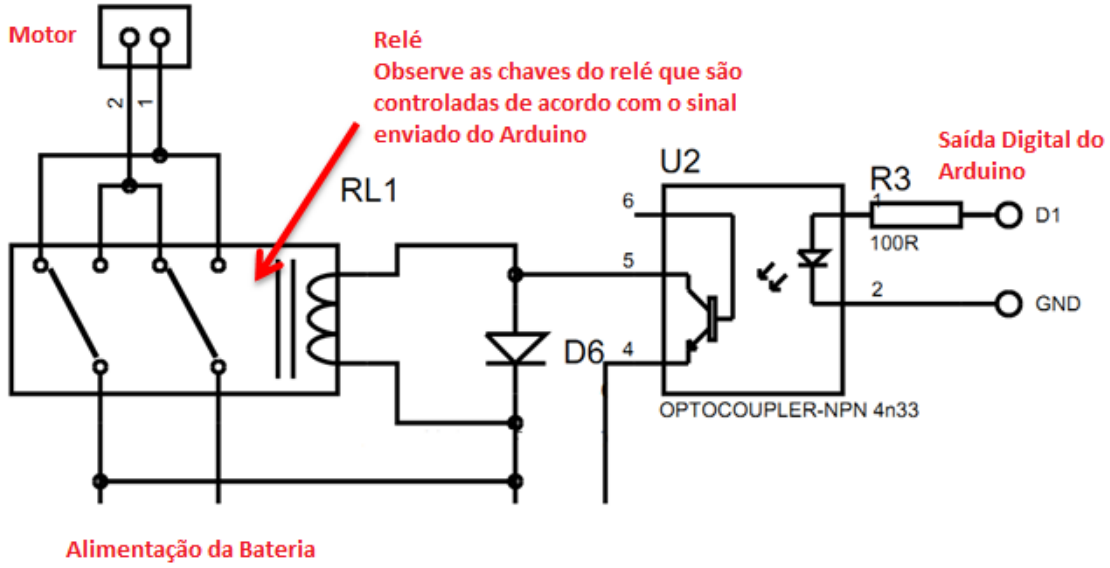
Quando montamos um circuito devemos pensar também em sua proteção, pois de um lado possuímos um sistema sensível (o Arduino), denominado **transmissor**, e do outro lado possuímos um sistema responsável pelo movimento de motores, denominado, **receptor**. Devemos realizar um isolamento dos circuitos para evitar possíveis problemas como fuga de corrente ou ligações e curtos não planejados. Para isso usamos um **Acoplador Óptico**, onde o sinal recebido do Arduino é transformado dentro do CI em emissão de um LED infravermelho que é identificado por um sensor e consequentemente enviando um sinal de saída, com isso, **isolamos totalmente o circuito transmissor do emissor** evitando possíveis problemas.

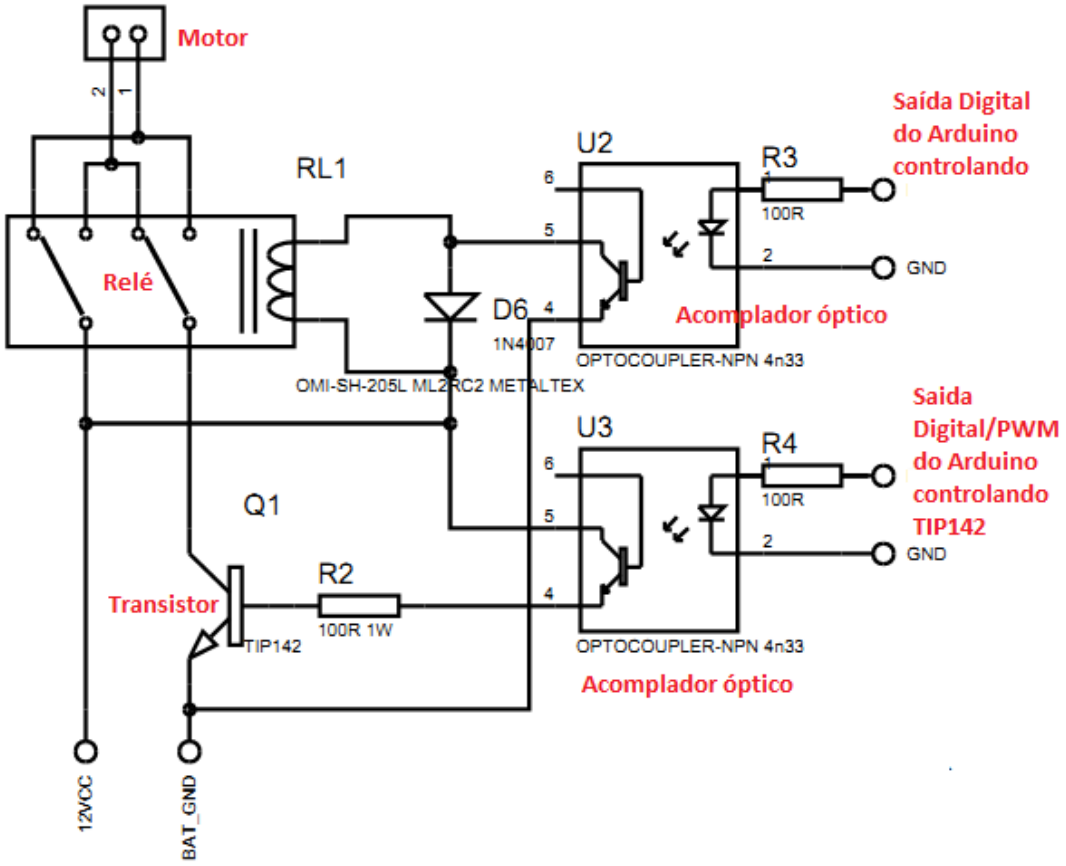


Se observarmos o circuito veremos após um acoplador óptico existe um transistor NPN TIP142 funcionando como uma chave. Quando o Arduino envia um sinal alto a este transistor, ele é ativado permitindo que a corrente vinda do motor flua até o GND da bateria usada, assim, fechando o circuito e fazendo o motor funcionar.



Conectado diretamente nos pinos dos motores são encontrados relés. Os relés funcionam como uma chave. Quando o Arduino enviar um pequeno sinal, o relé irá alterar as chaves, assim, alterando o sentido da corrente e fazendo o motor girar no sentido inverso.





PROGRAMAÇÃO PONTE H

Para programar uma Ponte H deve-se levar em consideração os conceitos apresentados sobre entrada e saída de dados. Esta programação leva o sistema de Ponte H apresentado anteriormente como referência. Esta programação possui velocidade constante. **Exemplo 10.**

```
#define relea 2 // definindo rele motor A
#define transa 3 // definindo transistor motor A
#define releb 4 // definindo relé motor B
#define transb 5 // definindo relé motor B
void setup(){
  // inicialização das saídas dos motores
  for (int i=2; i<6; i++){
    pinMode(i, OUTPUT);
  }
}
void setup(){
  frente();
  delay(2000);
```

```
  parar();
  delay(1000);
  paratras();
  delay(1000);
}

void frente(){
  //enviando sinal alto de cada componente de forma individual
  digitalWrite(transa, HIGH);
  digitalWrite(transb, HIGH);
}
void paratras(){
  // enviando sinal alto de TODOS componentes através da função for
  for(int i=2, i<6, i++){
    digitalWrite(i, HIGH);
  }
}
void parar(){
  digitalWrite(relea, LOW);
  digitalWrite(trana, LOW);
  digitalWrite(releb, LOW);
  digitalWrite(transb, LOW);
}
```

Observe na programação anterior que foram criadas funções específicas para cada um dos movimentos dos motores. O modo que os motores irão funcionar pode ser feito de diversas maneiras e utilizar várias funções especificadas anteriormente, porém, caba ao programador organizar o programa da maneira que desejar.

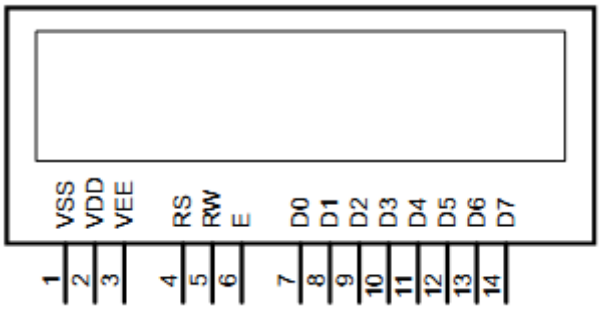
OBSERVAÇÃO!

*Caso você queira controlar a velocidade do eixo de um motor DC, você deve programar da mesma maneira que o **exemplo 8**, porém, invés de aplicar o sinal a um LED, você irá aplicar ao componente usado no seu sistema de controle.*

CONECTANDO UM DISPLAY DE CRISTAL LIQUIDO (LCD)

Conectar um display de cristal líquido (LCD) no Arduino não é uma tarefa tão difícil como parece. Para o Arduino funcionar com o LCD, ele dever ser baseado no chipset Hitachi ou similares. Vamos usar como referência um LCD alfanumérico de 16x2 (16 colunas, 2 linhas).

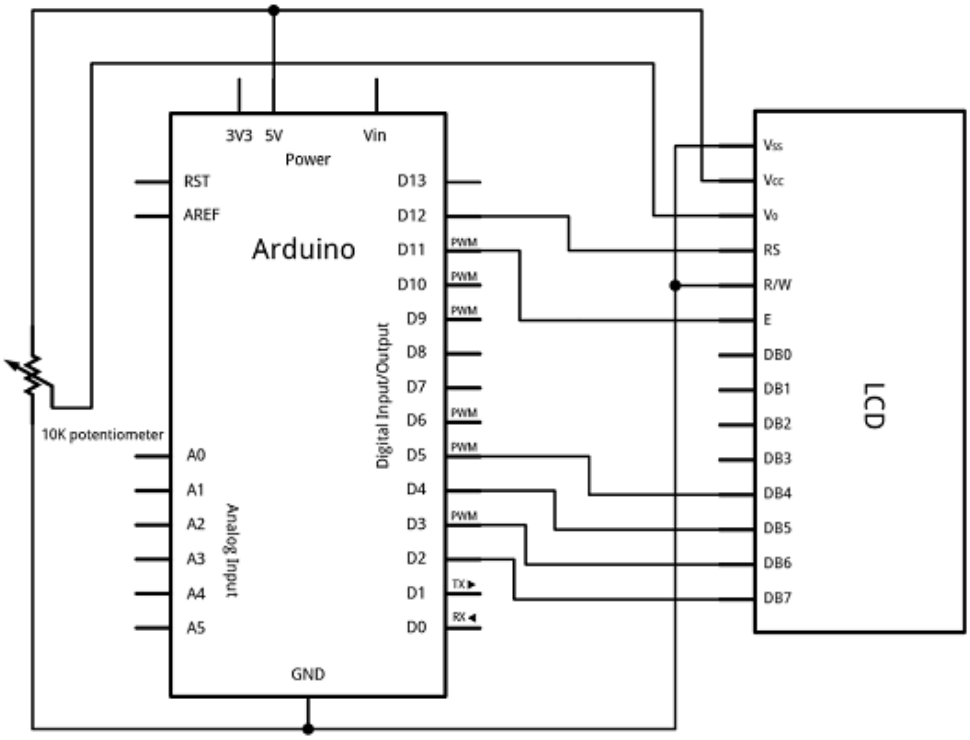
O LCD possui uma série de pinos que devem, ou não, ser conectados no Arduino, para isso, devemos entendê-los. Observe o desenho e as explicações abaixo para melhor compreender.



- **VSS:** Alimentação do display: Ground, Terra;
- **VDD**: Alimentação do display: 5 V;
- **VEE/VO:** Ajuste de contraste. Alterando a tensão aplicada neste pino, você altera o contraste do display;
- **RS (Register Select):** controla em qual parte do LCD você está enviando dados. Este pino controla o que será exibido na tela e o que são instruções de registro;
- **RW (Read/Write):** este pino seleciona se estará atuando em modo de leitura ou envio de dados;
- **E (Enable):** habilita o envio de dados aos registros;
- **D0 a D7:** controla o estado dos pinos (HIGH ou LOW)

Os displays LCD compatíveis com o chipset Hitachi podem atuar nos modos de 4-bits ou 8-bits, porém, no modo de 4-bits é necessário apenas 7 pinos I/O do Arduino, enquanto no modo 8-bits é necessário 11 pinos I/O. Se você for trabalhar apenas com textos o modo de 4-bits é suficiente.

Veja como fazer a ligação entre o display LCD e o Arduino corretamente.



Não é necessário usar resistores ou qualquer outro componente entre o Arduino e o LCD. Apenas é feito o uso de um potenciômetro para regular a tensão aplicada ao pino **VEE/VO** para controlar o contraste do monitor.

PROGRAMANDO UM LCD EM ARDUINO

Neste caso definimos as seguintes conexões:

:

Pino no LCD	Pino no Arduino
RS	12
E	11
D4	5
D5	4
D6	3
D7	2

Com isso podemos começar a realizar a programação baseada em uma biblioteca já contida no Arduino, a "**LiquidCrystal.h**".

Esta biblioteca é compatível com os displays baseados no chipset Hitachi e ela possui 20 diferentes funções. Com estas funções você possui controle total sobre seu display LCD. Em nossos estudos não iremos aprender todas estas funções, apenas as básicas para o funcionamento do LCD, caso você tenha interesse o site oficial do Arduino oferece um estudo completo sobre elas.

Para realizar a programação basicamente deve-se:

1. **Incluir a biblioteca do LCD:**
   A biblioteca baseada no chipset Hitachi já está contida no Arduino e pode ser acessível através do menu *Sketch* > *Import Library* > *LiquidCrystal.*
2. **Identificar cada um dos pinos conectados ao Arduino:** para realizar esta identificação usamos a seguinte função, onde devemos colocar o número do pino conectado ao Arduino com o respectivo nome.
   LiquidCrystal lcd(**RS**, **RW**, **E**, **D4**, **D5**, **D6**, **D7**);
   NOTA: *muitas vezes o uso do pino RW pode ser desconsiderado, assim, não sendo necessário conectá-lo ao Arduino ou identificando-o na programação.*
3. **Iniciar o display:** iniciamos o display com a seguinte função, onde identificamos o número de linhas e de colunas, neste caso, usamos um display 16x2:
   lcd.begin(*COLUNAS*, *LINHAS*);
   Ex: **lcd.begin**(16,2);
4. **Imprimir dados no display:** para imprimir dados, podemos usar a seguinte função:
   lcd.print("**Mensagem**");

Observe o **exemplo 11**.

```
/*Este exemplo de código é de domínio público */
#include <LiquidCrystal.h> // inclui o código da biblioteca
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // inicializando biblioteca e os respectivos pinos
void setup() {
 lcd.begin(16, 2);  // inicializa o display com o número de colunas e linhas adequado
 lcd.print("hello, world!");  // Imprimindo uma mensagem no LCD
}
void loop() {
 // Fixa o cursor na coluna 0, linha 1
 // NOTA: a contagem das linhas começa em 0, portanto a linha 1 é a segunda linha
 lcd.setCursor(0, 1);
 // Imprime o número de segundos desde o reset.
 lcd.print(millis()/1000);
 }
```

USO DE SENSORES

Um sensor é um dispositivo que responde com um sinal elétrico a um estímulo físico de uma maneira específica. Como estes sinais são uma forma de energia, os sensores podem ser classificados de acordo com o tipo de energia que eles detectam, ou seja, podem ser sensores:

- De luz: células solares, fototransistores, fotodiodos;
- De som: microfones, sensores sísmicos;
- De temperatura: termômetros, termístores (resistores sensíveis à temperatura), termostato;
- De calor: calorímetro;
- De radiação: dosímetro, contator Gaiger;
- De resistência elétrica: ohmímetro;
- De corrente elétrica: galvanômetro, amperímetro;
- De tensão elétrica: voltímetro;
- De potência elétrica: wattímetro;
- Magnéticos: magnetômetro, compasso magnético;
- De pressão: barômetro;
- De movimento: velocímetro, tacômetro;
- Mecânico: sensor de posição;
- De proximidade;
- De distância.

Como foi dito, os sensores transformam um estímulo físico em tensão elétrica, assim dizemos que então o sinal é enviado ao sistema lógico, neste caso, o Arduino. Há sensores analógicos e também digitais, quando montarmos algum sistema devemos ter atenção ao tipo de sensor que usamos e as suas características básicas como corrente, tensão, possíveis usos, etc.

Devemos ter uma atenção especial na parte da programação. Para programar o sensor corretamente necessitamos realizar testes e verificar os resultados. Qualquer programador pode estudar como o sensor age e então realizar uma programação específica e própria para ele, porém, isso acaba ocupando muito tempo e o resultado pode não ser tão significativo, por isso, o uso de sensores com o Arduino pode ser facilitado. Hoje grande parte dos fabricantes de diversos sensores em seus sites oficiais, em seus datasheets e em fóruns na internet oferecem informações específicas para o uso com Arduino e até fornecem bibliotecas já prontas, assim facilitando a programação e podendo fazer o sensor funcionar corretamente com poucas linhas de códigos.

Por estes motivos não se pode ensinar uma forma de programação válida para funcionar com qualquer sensor, é necessário colocar em prática todos os ensinamentos que foram passados até aqui como a leitura e escrita de dados analógicos e digitais, o uso de variáveis para armazenar as informações, o uso de ondas PWM, funções, estruturas de controle de fluxo, etc., enfim, o que vai fazer o sensor funcionar corretamente é o conhecimento que você adquiriu ao longo das suas experiências.

Ao longo da apostila, temos utilizado principalmente em exemplos o uso de LEDs e de potenciômetros, isso facilita pois estes elementos funcionam como a base de conhecimento necessária para funcionar com sensores. Tanto potenciômetros como sensores analógicos enviaram um sinal de 0 a 1023 e cabe ao programador mostrar o que cada período neste intervalo significa. A leitura do sinal de um LED ou de um botão, por exemplo, funciona tão bem para demonstrar a leitura de um dado analógico assim como em sensores digitais.

Nesta apostila onde falamos sobre ondas PWM (página 30), possuímos dois exemplos que poderiam ser usados com sensores, sendo um dos exemplos de um sensor ultrassônico, onde ele capta a distância de objetos em sua frente num raio de distância específico.

O Arduino pode captar as informações de um sensor analógico através da função *analog**Read()*** e pode enviar informações ao sensor através da função *analog**Write()***. Sensores analógicos recebem um sinal de 0 a 1023 e cabe ao programador designar a grandeza medida. Exemplo, você possui um sensor de temperatura analógico que mede grandeza em célsius de 0°C a 50°C, dizemos que o sinal 0 equivale a 0°C e 1023 equivale a 50°C, cabendo ao programador desvendar os valores equivalentes entre estes resultados.

Sensores digitais funcionam apenas com sinal alto (**HIGH**) ou sinal baixo (**LOW**). Uma chave de fim de curso, por exemplo, pode ser considerado um sensor digital. Dizemos que a chave é considerada como NF (normalmente fechada), assim, se for feita uma leitura através de *digital**Read()*** quando a chave não estiver pressionada obteremos o valor 1 (HIGH, sinal alto), caso a chave seja pressionada, obteremos o valor 0 (LOW, sinal baixo), assim, cabe ao programador decidir qual será a utilidade do uso deste sensor.

Porém, há sensores de uso mais complexo, onde você pode obter dados mais específicos ou variados tipos de dados, para isso, o fornecedor sempre oferece um manual de como realizar a programação e de como usar as bibliotecas dos sensores, mas para isso, você sempre deverá estar procurando por estas informações em fóruns específicos e no site dos fabricantes.

ANEXOS

1. Sites de fabricantes e vendedores de Arduinos originais, de plataformas baseadas em Arduino ou de shields.

1.1. http://arduino.cc/ - Site Oficial do Arduino;
1.2. http://www.ladyada.net/make/boarduino/ - Boarduino, plataforma baseada em Arduino;
1.3. http://www.seeedstudio.com/blog/2008/09/15/seeeduino-release-notice-ver10b/ - Seeduino, baseado em Arduino;
1.4. http://shieldlist.org/ - Lista de todos os fabricantes e shields para Arduino existentes no mercado;
1.5. http://www.multilogica-shop.com/ - vendedor oficial de produtos Arduino no Brasil
1.6. http://www.sparkfun.com/ - vendedor oficial internacional de Arduino, shields e sensores;
1.7. http://www.adafruit.com/ - vendedor oficial internacional de Arduino, shields e sensores.

2. Sites de fóruns e outros sites onde fornece informações importantes sobre o Arduino.

2.1. http://www.arduino.cc/forum/ - Fórum oficial Arduino.
2.2. http://www.ladyada.net/ - Portfolio de informações sobre Arduino e seus usos;
2.3. http://forums.adafruit.com/viewforum.php?f=37&sid=e1903ded7b393d4f0c1619914c76a6af – Fórum Adafruit sobre Arduino;

BIBLIOGRAFIA

- Site Oficial Arduino: http://arduino.cc/ - Acesso em 05 de Outubro de 2011;
- BANZI, Massimo; Getting Started With Arduino; O'Reilly Media Inc.; 2010;
- MCROBERTS, Michael; Beginning Arduino; Apress; 2010;
- SANTOS, Nuno Pessanha; Arduino, Introdução e Recursos Avançados; 2009; Disponível em http://pt.scribd.com/doc/57058743/Tutorial-Arduino;
- Apostila Introdução a Linguagem C; Unicamp; Disponível em ftp://ftp.unicamp.br/pub/apoio/treinamentos/linguagens/c.pdf.