



KIT AVANÇADO
PARA **ARDUINO**
V3.0

Parabéns por adquirir o Kit Avançado para Arduino da RoboCore!

Este material é composto por nove tópicos, indicados para usuários que **já** tem contato com a plataforma Arduino e conhecem os componentes eletrônicos básicos. Em cada tópico existem alguns experimentos. Os usuários interessados em seguir este manual necessitam ter algum conhecimento prévio em linguagem C++ (a linguagem do Arduino). Caso você não tenha nenhum conhecimento em Arduino, componentes e programação em linguagem C++, indicamos que adquira o [Kit Iniciante para Arduino](#) da RoboCore. Mais detalhes no site www.RoboCore.net.

Por se tratar de um kit indicado a usuários com conhecimento prévio, você notará que não existem explicações aprofundadas em artigos básicos, como a função de componentes básicos e/ou sintaxe de programação básica. Caso você tenha sido usuário do Kit Iniciante para Arduino da RoboCore, verá uma diferença no que diz respeito à diagramação dos circuitos. Os esquemas serão apresentados de uma forma diferente, com símbolos conhecidos em esquemas eletrônicos. Para estes experimentos, será necessário o uso de uma protoboard – **não inclusa no kit** (por supomos que o usuário deste kit já tenha conhecimentos em Arduino e já tenha feito uso do Kit Iniciante para Arduino, supomos que o usuário já possua uma placa Arduino e uma protoboard para testes).

Veja abaixo a lista de tópicos existentes neste material:

- **Servo Motor** pág. 04
Componentes: 01x Servo Motor
Descrição: Aprenda conceitos básicos de movimentação de servo motores, além de ver como você pode modificar seu servo motor de movimento angular para servo motor de rotação contínua.
- **Acionamento de Motores** pág. 07
Componentes: 01x Motor DC + 01x TIP122 + 01x Potenciômetro + 01x Barra de LEDs
Descrição: Aprenda como funcionam motores DC e como controlá-los através de uma placa Arduino. Veja também a intensidade do motor através da barra de LEDs.
- **Utilizando LEDs Infravermelhos** pág. 16
Componentes: 01x LED Receptor Infravermelho + 01x LED Emissor Infravermelho
Descrição: Aprenda a fazer barreiras invisíveis a olho nu.
- **Teclado de 12 Botões** pág. 19
Componentes: 01x Teclado com 12 botões + 01x Servo Motor
Descrição: Aprenda a usar um teclado com 12 botões, muito utilizado em telefones e em dispositivos eletrônicos onde você deve colocar uma senha para acessar um recinto.
- **Circuito Integrado 555** pág. 25
Componentes: 01x Circuito Integrado 555
Descrição: Veja algumas, das centenas de possibilidades de projetos, que este pequeno circuito integrado serve. Neste experimento utilizaremos o software [Processing](#).
- **Acionamento de Cargas com Módulo Relé** pág. 36
Componentes: 01x Módulo Relé
Descrição: Aprenda como acionar cargas de maior porte. Com este experimento será possível acionar e desacionar qualquer carga cuja especificação esteja dentro das especificações do relé presente no módulo.
- **Controle Remoto** pág. 39
Componentes: 01x Controle Remoto + 01x LED Receptor Infravermelho + 01x Módulo Relé
Descrição: Aprenda como utilizar a maioria dos controles remotos com Arduino, e no final aprenda a ligar e desligar uma carga usando o controle.

- **Acelerômetro**

Componentes: 01x Acelerômetro + 02x Resistores de 300Ω

pág. 43

Descrição: Aprenda como funciona este sensor, que está presente na maioria dos dispositivos móveis da atualidade, como smartphones, tablets e até notebooks.

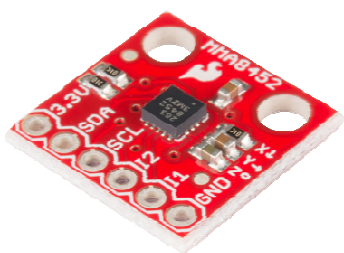
- **Sensor Ultrassônico**

Componentes: 01x Sensor Ultrassônico

pág. 51

Descrição: Aprenda como funciona este sensor, que consegue fazer leituras precisas de distância, com resolução de 1cm.

Veja abaixo alguns itens que acompanham este kit:



Acelerômetro MMA8452Q



Controle Remoto



Micro Servo Motor



Sensor Ultrassônico HC-SR04



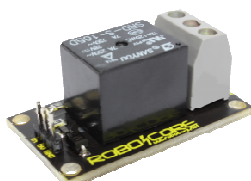
Transistor TIP122



Resistores e Componentes Diversos



Motor DC



Módulo Relé



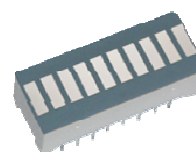
LEDs Infravermelhos



Teclado 12 Botões



Circuito Integrado (CI) 555



Barra Gráfica de LEDs

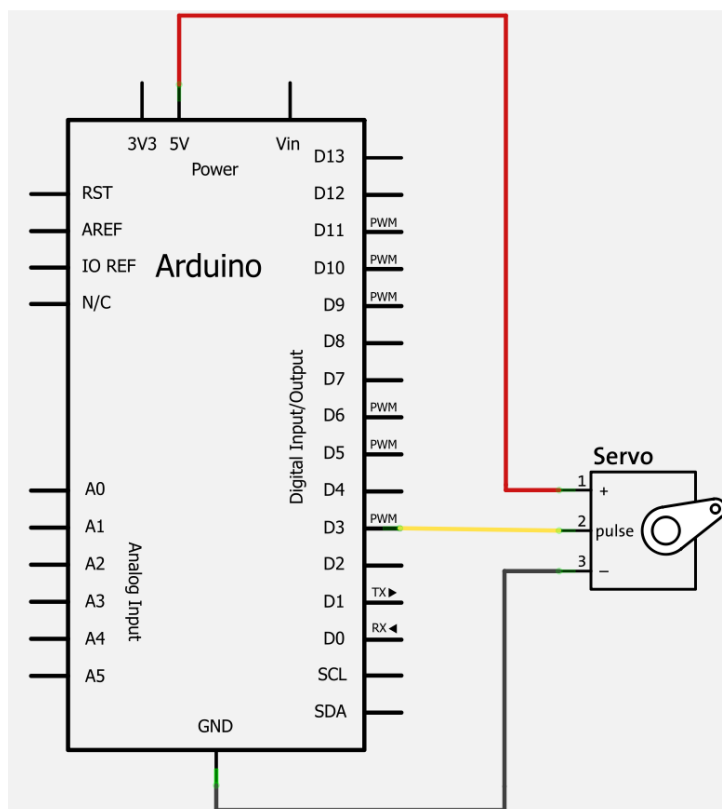
- **Servo Motor**

Componentes: 01x Servo Motor

Descrição: Aprenda conceitos básicos de movimentação de servo motores, além de ver como você pode modificar seu servo motor de movimento angular para servo motor de rotação contínua.

Neste experimento, utilizaremos a biblioteca **Servo** (inclusa na IDE do Arduino). Bibliotecas nada mais são do que um conjunto de funções prontas para serem utilizadas pelo desenvolvedor. Atualmente, como milhões de pessoas estão utilizando as placas Arduino ao redor do mundo, você pode encontrar milhões de bibliotecas diferentes, para as mais diversas funções. A biblioteca **servo** permite que, com apenas um comando simples você consiga mover o eixo de seu servo motor. O servo motor que acompanha o kit é um servo motor de posição angular. Ou seja, o curso deste servo motor é de - aproximadamente - 0° a 180° (na prática os servos não costumam chegar a ter toda esta abertura), sendo 90° o ângulo que corresponde ao meio do curso. Existem também servos de rotação contínua. Ao final deste experimento será mostrado como fazer esta alteração. Trata-se de um processo trabalhoso. Um servo motor de rotação contínua não possui fim de curso, podendo seu eixo girar bem como um motor convencional. Quando um servo motor é de rotação contínua, a posição 90°, que antes era o meio do curso, agora representa a “velocidade” zero, ou seja, motor parado. Para que o servo gire seu eixo para um lado com toda velocidade, basta acioná-lo a 180°. Para que o servo gire seu eixo para o outro lado com toda velocidade, basta acioná-lo a 0°. Vale dizer que, por se tratar de um servo motor sua velocidade final não é alta.

Veja abaixo o esquema de ligação do servo no Arduino para este experimento:



Os esquemas de montagem neste material serão apresentados desta forma. Apenas para título de explicação neste experimento, você deve ligar o fio vermelho do servo no 5V da placa Arduino, o fio marrom no GND e o fio laranja na porta **digital 3**. Ligaremos na porta 3, pois é uma porta digital com função PWM, função compreendida no manual do Kit Iniciante para Arduino.

Após feita esta ligação, um código inicial apenas para ver algo acontecer com o eixo de seu servo motor pode ser o seguinte:

```
/******\
** ROBOCORE KIT AVANÇADO PARA ARDUINO **
** Servo Motor 1 **
\*****/

#include <Servo.h> //Este comando inclui a biblioteca Servo neste código

Servo Servo1; //Identifica um objeto do tipo Servo chamado Servo1

void setup()
{
  Servo1.attach(3); //Pino onde o servo deve estar colocado
}

void loop() {
  Servo1.write(90); //Posição em graus do eixo do servo
}
```

O código acima é muito simples, e pouco funcional, pois ele simplesmente faz o eixo do servo motor ir até a posição 90° e por lá ficar. Você pode testar trocar o valor 90 para quaisquer valores entre 0 e 180. Você verá que o eixo do servo irá rodar até a posição desejada. A fim de tornar este experimento mais dinâmico, você pode compilar o código de exemplo da biblioteca Servo chamado Knob. Para tanto você irá precisar de um potenciômetro e precisará fazer as ligações conforme informado na descrição do experimento.

Vamos agora fazer um programa um pouco mais interativo para mexer o eixo deste servo. Grave o seguinte programa em seu Arduino:

```
/******\
** ROBOCORE KIT AVANÇADO PARA ARDUINO **
** Servo Motor 2 **
\*****/

#include <Servo.h> //Este comando inclui a biblioteca Servo neste código

Servo Servo1; //Identifica um objeto do tipo Servo chamado Servo1
int Recebido; //Variável que armazenará o valor recebido pela serial
int posicao; //Variável que armazenará as posições do servo

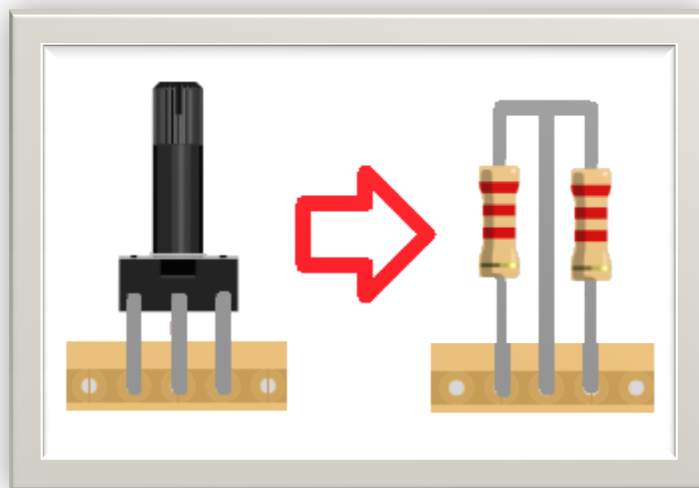
void setup(){
  Servo1.attach(3); //Pino onde o servo deve estar colocado
  Serial.begin(9600); //Inicia a comunicação serial com taxa de 9600
}

void loop(){
  if (Serial.available() > 0) { //Se algo vier pela porta serial...
    Recebido = Serial.read(); //O que receber, armazenar na variável Recebido
    if (Recebido == '+'){ //se receber o sinal de + pela serial faça:
      posicao = posicao + 10; //adicione 10 à variável incremento
      Serial.println(posicao); //mostra no Monitor Serial o valor da posição
    }
    if (Recebido == '-'){ //se receber o sinal de - pela serial faça:
      posicao = posicao - 10; //subtraia 10 à variável incremento
      Serial.println(posicao); //mostra no Monitor Serial o valor da posição
    }
    Servo1.write(posicao); //Escreve a posição no servo
  }
}
```

O código acima faz com que, cada vez que chegar um símbolo de positivo (+) pela porta serial, o servo aumente sua posição em 10° e, cada vez que chegar um símbolo de negativo (-), o servo diminua sua posição em 10°. Você pode mudar este passo para o número que quiser para ter mais ou menos resolução na movimentação do eixo do servo. Para enviar um símbolo de positivo ou negativo para sua placa Arduino, basta abrir o Monitor Serial (o botão na IDE do Arduino cujo ícone representa um uma lupa). Com o Monitor Serial aberto, selecione no canto inferior direito a **taxa de comunicação de 9600**, que significa que a placa irá enviar/receber dados a uma taxa de 9600bps. Então digite um sinal de positivo e clique em Send (ou aperte a tecla enter). Você verá o eixo do servo se mover em um sentido e poderá ler na tela do computador via monitor serial em qual ângulo está o eixo do servo. Caso nada aconteça, feche a tela do monitor serial e reabra.

Servo motores são muito utilizados em braços robóticos, mãos robóticas, robôs humanóides, ponte-H mecânica com switches, etc, devido a sua facilidade e precisão nos movimentos. Porém, também é possível utilizar servo motores como motores para um robô se mover com rodas. Para isto é necessário adquirir um Servo Motor de **Rotação Contínua**, ou modificar um servo para esta função.

Para modificar um servo comum para rotação contínua, deve-se retirar os 04 parafusos na parte traseira de seu servo e retirar todos os componentes de dentro do mesmo. Tome cuidado para não perder as engrenagens da parte superior e sua ordem de instalação. Corte com um alicate a trava de plástico na engrenagem sempre com cuidado para não comprometer a engrenagem em si. O próximo passo é retirar o potenciômetro existente dentro do servo e substituí-lo por dois resistores de mesmo valor. O mais indicado é utilizar dois resistores de 2,2kΩ. Esta substituição serve para “enganar” o servo, pois a partir de agora ele sempre “pensará” que estará na posição central (pois não há mais referência de resistência – que era fornecida pelo potenciômetro) e passará a se comportar como um servo de rotação contínua, onde 180 agora representa a velocidade total para um lado e 0 representa a velocidade total para o outro lado, sendo 90 a posição de parado. Este processo parece simples, porém você deve estar acostumado com circuitos eletrônicos para fazer tal modificação. Para ilustrar, você deverá fazer a troca do potenciômetro pelos resistores segundo a imagem abaixo:



ATENÇÃO:

Ao modificar um servo normal para rotação contínua, não será possível modifica-lo novamente para sua condição original de fábrica. As modificações são permanentes. Caso você não saiba o que está fazendo, não tente modificar seu servo pois, provavelmente, você irá inutiliza-lo.

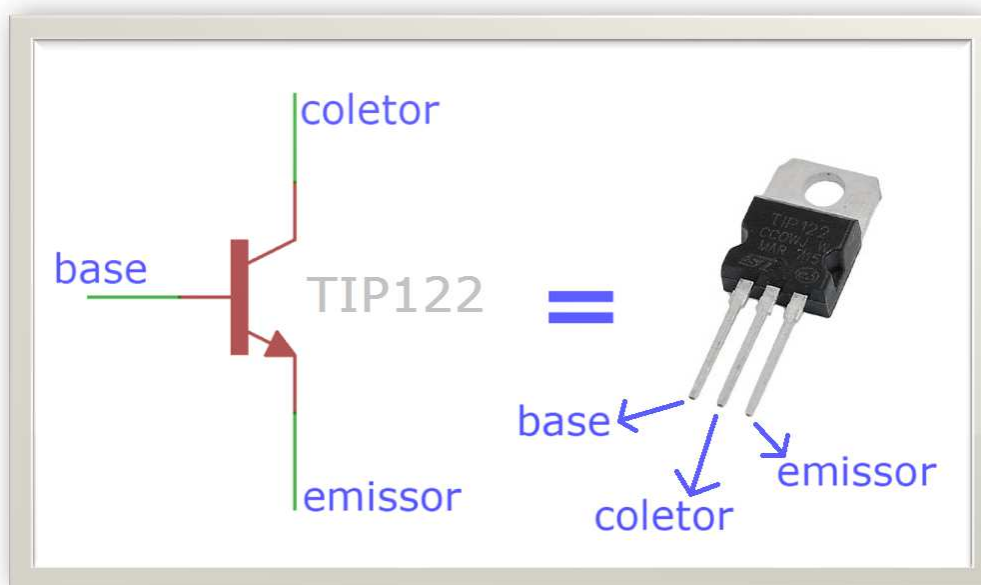
- **Acionamento de Motores**

Componentes: 01x Motor DC + 01x TIP122 + 01x Potenciômetro + 01x Barra de LEDs

Descrição: Aprenda como funcionam motores DC e como controlá-los através de uma placa Arduino. Veja também a intensidade do motor através da barra de LEDs.

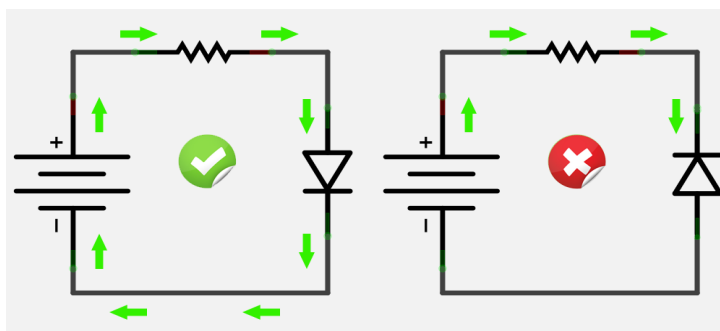
Brincar com motores é sempre muito legal, pois as possibilidades são muitas. Motores são usados em aplicações fixas e móveis. São encontrados em processos de fabricação (em esteiras, por exemplo), em sistemas de locomoção de robôs, cadeiras de roda motorizada, aparelhos de DVD, além de muitos outros projetos. No experimento deste tópico, iremos utilizar um motor de corrente contínua com escovas, ou mais brevemente, Motor DC. Este é apenas um dos tipos de motores existentes nas aplicações. Também existem os servo motores (já vistos neste material), motores de passo, motores brushless, motores de corrente alternada, motores trifásicos e motores universais. Após o servo-motor, o motor DC é o motor mais fácil de ser controlado, porém diferente do servo, motores DC não podem ser ligados diretamente a uma placa Arduino por precisarem de uma corrente elétrica maior do que a porta do Arduino pode fornecer. As portas digitais do Arduino podem fornecer no máximo **40mA** de corrente e um motor, mesmo que pequeno, consumirá mais do que isso, ainda mais se for submetido a algum tipo de carga - carga esta que pode ser, por exemplo, uma roda em seu eixo encostada ao chão para fazer o movimento de um robô. Ao mexer com motores DC, deve-se ter em mente que a corrente quando não há nada em seu eixo é uma, e quando há uma roda em contato com o chão é outra (necessariamente maior). Então como fazer para controlar um motor DC com uma placa Arduino sem ligá-lo diretamente nas portas do Arduino? Iremos usar agora um **transistor**!

Um transistor é um componente que possui diversas funções na eletrônica, desde fazer lógica por meio de componentes discretos, até servir como chave eletrônica. De fato, a maioria dos chips que usamos nada mais são que um "mar" de transistores. No nosso caso, iremos usá-lo como uma chave, onde sempre que houver um sinal em sua base, ele deixará a corrente fluir entre o coletor e o emissor. Estes conceitos ficarão mais claros vendo a imagem a seguir:

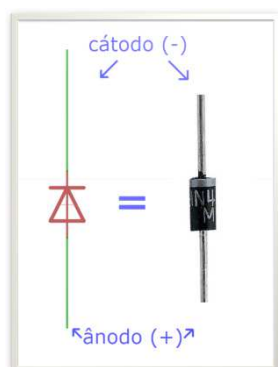


Para saber mais sobre o TIP122, aconselhamos ler o datasheet do componente.

Além de conhecer o transistor, também precisamos conhecer um outro componente que nos ajudará neste experimento: o **diodo**. O diodo é um componente que deixa a corrente elétrica passar somente em um sentido. Este componente tem dois terminais, um é conhecido como ânodo e o outro é conhecido como cátodo. Veja na figura abaixo como é o componente e como ele se comporta:



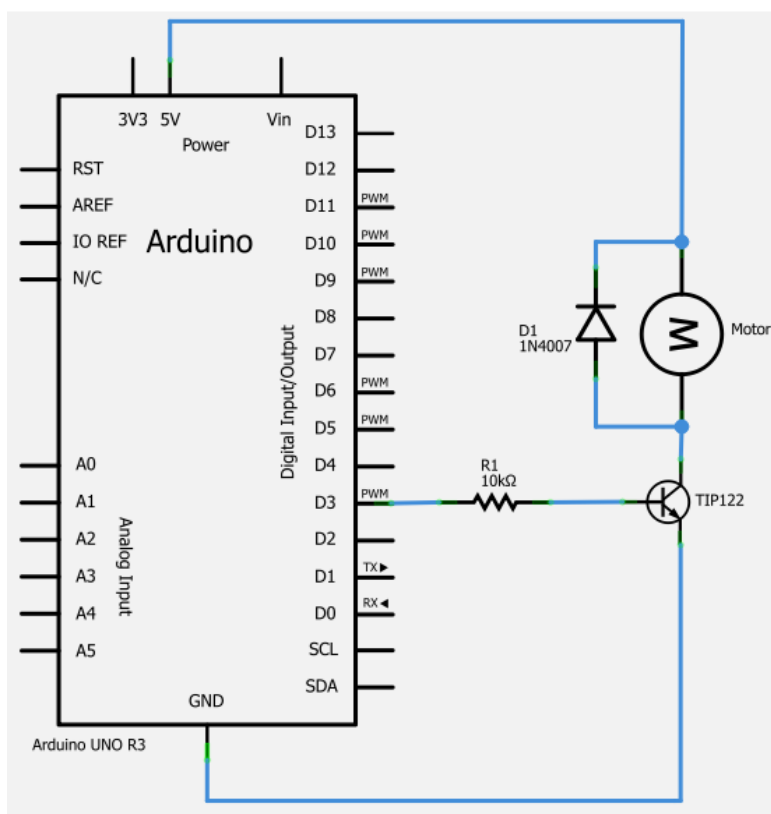
Circuito à esquerda: diodo deixa a corrente passar
Circuito à direita: diodo não deixa a corrente passar



Ou seja, ele serve para barrar a corrente do circuito. No circuito que iremos montar neste experimento ele servirá para prevenir correntes parasitas. Esta corrente parasita nada mais é do que uma corrente indesejável de alto valor (que pode queimar o restante do circuito), que pode ser gerada quando acionamos muito rapidamente, ou paramos rapidamente, nosso motor.

Tome cuidado que este componente possui polaridade.
Veja na figura ao lado onde é o anodo e onde é o catodo do diodo.

Para nosso primeiro experimento da parte de motores, encaixe dois jumpers nas duas terminações traseiras do motor, e monte o seguinte circuito:



Salve então em seu Arduino o seguinte código:

```
const int pwm = 3;

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

void loop(){
  for(int x = 0; x <= 255; x++){
    analogWrite(pwm, x);
    delay(50);
    Serial.println(x);
  }

  for(int x = 255; x >= 0; x--){
    analogWrite(pwm, x);
    delay(50);
    Serial.println(x);
  }
}
```

Este programa é muito simples, ele simplesmente diz que chamaremos o pino 3 digital do Arduino de **pwm**, setamos ele como saída (bem como iniciamos a comunicação serial) no bloco setup, e no loop principal vamos incrementando e decrementando a variável **x** com intervalos de 50 milissegundos. Como a variável **x** é **local** nos dois loops *for*, eu posso criá-la duas vezes que uma não vai interferir na outra (quando acaba o *for*, a variável simplesmente deixa de existir). Você consegue entender o que está havendo e porque estamos indo apenas até 255?

O controle de velocidade do motor é feito através da técnica de modulação por largura de pulso, ou **PWM**. A resolução das placas Arduino permite que tenhamos valores de 0 a 255 (como já foi mostrado na apostila do Kit Iniciante para Arduino da RoboCore). Portanto, devemos entender que colocando um valor de pwm igual a **0** no motor, teremos ele parado; e colocando um valor de

255 teremos o motor na máxima velocidade. Na programação isto é gradual. Na primeira rotina *for* nós temos um loop que incrementa a variável **x** até seu valor máximo, que neste caso é 255. Para entender melhor a rotina *for*, vamos ver abaixo sua estrutura:

```
for ( int x = 0; x <=255; x ++ ){  
    //faz o que tem que ser feito neste laço de for até x ser menor ou igual a 255  
}
```

Para usar o **for** precisamos de 3 argumentos:

1 - **Inicialização de variável** (**int x = 0**). Podemos iniciar esta variável dentro do próprio parênteses, e ela somente será usada neste laço de *for* (por isso podemos declará-la novamente no laço de **for** seguinte).

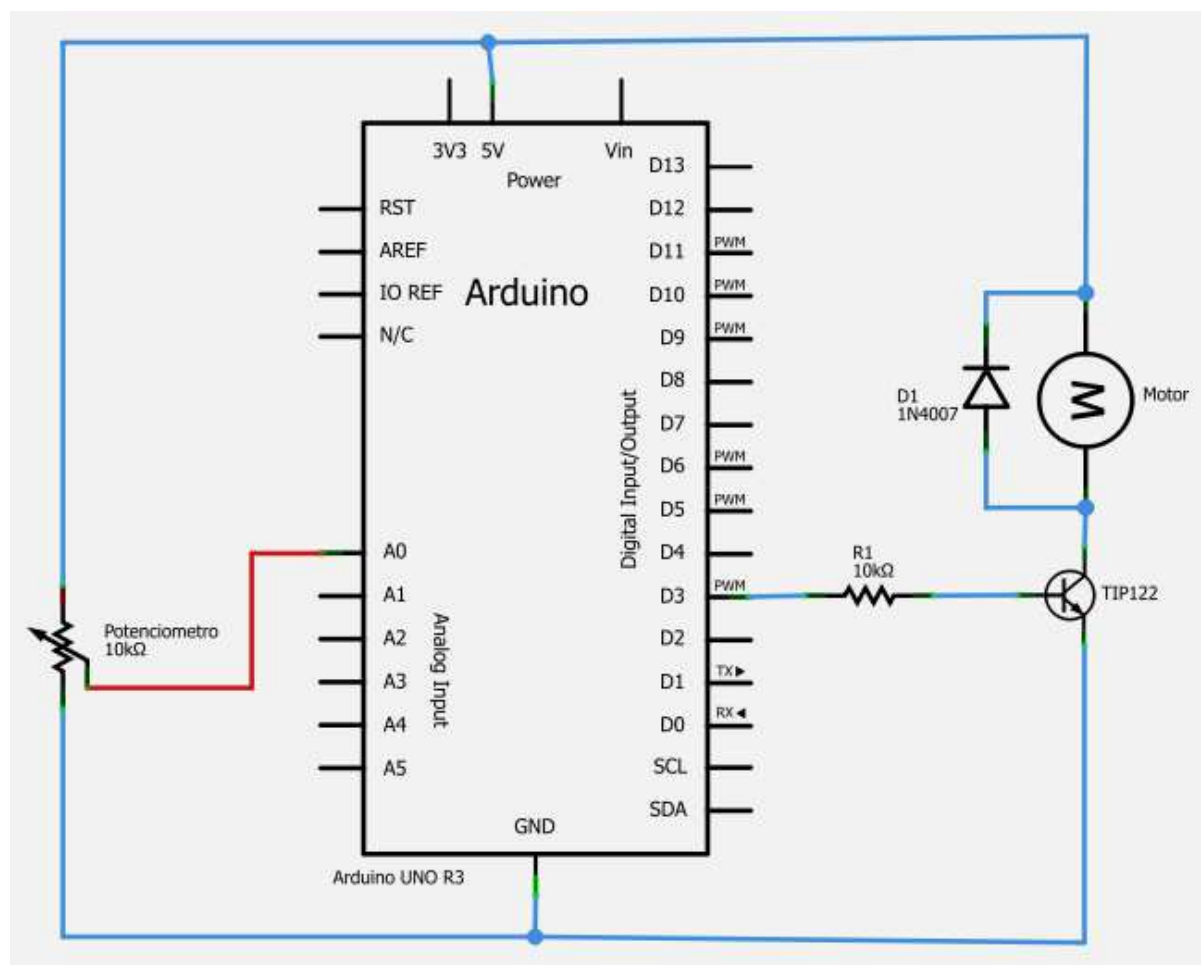
2 - **Condição da variável** (**x <= 255**). Este será o teste de condição do laço **for**. A rotina fica no loop até que o valor da variável supere a condição. Podemos entender a condição como **x** tendo que ter seu valor até que seja menor ou igual **255**. Quando **x** for maior que **255**, acaba o *for*.

3 - **Incremento da variável** (**x++**). O incremento neste caso será de **1 em 1**, pois quando usamos a variável seguida de dois sinais de mais na linguagem C++, queremos dizer o mesmo que: **x = x + 1**. Isto facilita muito a programação quando temos experiência. O mesmo ocorre para **x--** (que decrementa 1).

Dentro dos laços *for* temos a escrita analógica no pino **pwm**, que no caso é nosso pino digital 3. O valor a ser escrito varia de 0 a 255 com intervalos de 50 milissegundos. Isto faz com que nosso motor vá aumentando a velocidade gradualmente e, quando escrevermos 255 nele, ou seja, quando ele estiver na rotação **máxima**, ele começa a desacelerar até 0. Além disso, poderemos abrir o Monitor Serial enquanto nosso motor está em operação e ver qual a intensidade do mesmo, acompanhando o valor que está sendo usado no pino de PWM.

Nosso primeiro experimento com motores fez a placa Arduino controlar o motor através de uma inteligência própria. Isto é muito usado em robôs autônomos, onde a partir da leitura de sensores, o Arduino envia sinais para os motores dizendo se ele deve prosseguir ou parar, ou virar para algum dos lados.

Vamos agora controlar manualmente o motor. Para isto, usaremos um potenciômetro de 10k. Este potenciômetro servirá como um acelerador para o motor. Poderemos com ele aumentar ou diminuir a velocidade do motor apenas girando seu eixo. Para isto, inclua no circuito anterior o potenciômetro da seguinte forma:



Como visto anteriormente, o potenciômetro possui 3 pinos. Ele não possui polaridade, porém o pino do centro deve estar necessariamente ligado ao pino de entrada analógica 0 do Arduino. Os das extremidades devem um estar ligado ao GND e o outro ligado ao 5V.

Como podemos ver, a alteração do circuito é bem simples, mas será que a alteração do código é tão simples quanto? Coloque o seguinte código em sua placa Arduino:

```
const int pwm = 3;
const int pot = A0;
int ValorLidoPot_Real = 0;
int ValorLidoPot_Mapeado = 0;

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

void loop(){
  ValorLidoPot_Real = analogRead(pot);
  ValorLidoPot_Mapeado = map(ValorLidoPot_Real, 0, 1023, 0, 255);
  analogWrite(pwm, ValorLidoPot_Mapeado);
  Serial.println(ValorLidoPot_Mapeado);
}
```

Declaramos 3 novas variáveis. Primeiro continuamos dizendo que o pino 3 é nosso pino de pwm, o que irá controlar o motor. Depois dizemos que temos um potenciômetro ligado ao pino de entrada analógica 0, ou **A0**. Então declaramos duas variáveis cujo valor será alterado conforme o

código for rodando na placa, as variáveis **ValorLidoPot_Real** e **ValorLidoPot_Mapeado**. Estas duas variáveis irão colher os dados vindos do potenciômetro, porém com uma diferença: os limites máximo e mínimo de cada uma.

Uma placa Arduino possui um conversor analógico-digital de 10 bits. Isto quer dizer que a resolução máxima de valores analógicos que podemos ter, vai de 0 a 1023 (total de 1024 valores). Estes 1024 valores são encontrados a partir de uma rápida conta: $2^{10} = 1024$. Existem microcontroladores que possuem um conversor ADC (*analog-to-digital converter*) de 12 bits (como a placa Arduino DUE). Neste caso, ao invés dos valores irem de 0 a 1023, eles vão de 0 a 4095 (uma resolução muito maior). Para nós, nestes experimentos, um conversor de 10 bits de resolução está de bom tamanho.

Observando o código, vemos que a variável **ValorLidoPot_Real** recebe os valores diretos do potenciômetro, através da instrução `analogRead( )`. Portanto, os valores da variável **ValorLidoPot_Real** vão de 0 a 1023. Porém, nossa saída PWM vai apenas de 0 a 255! E agora, como adequar os valores para esta nova faixa de medição?

Felizmente, podemos usar no Arduino uma instrução chamada `map( )`. Esta instrução permite o mapeamento de valores, ou seja, transformar os limites de valores de determinada variável. É aí que surge nossa variável **ValorLidoPot_Mapeado**. Ela usa o comando **map** para trocar os limites da variável **ValorLidoPot_Real** para valores de 0 a 255. Veja como é a estrutura desta instrução:

```
map ( variável_que_precisa_ser_mudada , valor_mínimo_real ,  
valor_máximo_real , valor_mínimo_mapeado , valor_máximo_mapeado ).
```

Veja que as **cinco** instruções devem ser separadas por **vírgula**. Assimilamos uma nova variável para este mapeamento, por isso temos a linha de comando:

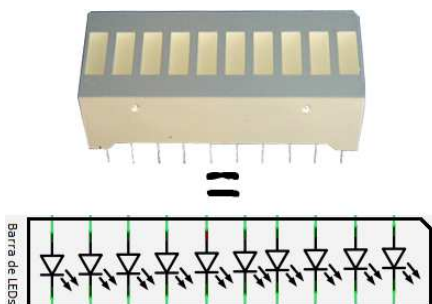
```
ValorLidoPot_Mapeado = map(ValorLidoPot_Real, 0, 1023, 0, 255);
```

Agora sim! Grave o código no Arduino e gire o eixo do potenciômetro. Isto fará com que o motor gire devagar e rápido. Os valores da intensidade do motor continuam podendo ser vistos através do Monitor Serial do programa do Arduino.

NOTA: Veja que assim que você começa a rodar o eixo do potenciômetro, o motor não começa a rodar seu eixo imediatamente. Isto ocorre porque o motor é algo mecânico, para rodar seu eixo ele precisa vencer a inércia que o deixa parado. Após vencer esta inércia, seu eixo roda e então a mudança de velocidade se torna imediata ao girar o potenciômetro.

Agora vamos supor que temos uma aplicação onde é necessário estar sempre de olho em qual é a intensidade que o motor está trabalhando. Vamos supor não ter um computador por perto, portanto não poderíamos utilizar o Monitor Serial. O que podemos fazer para verificar, de uma forma fácil, a intensidade do motor em tempo real sem usar um computador? Vamos usar a barra de LEDs!

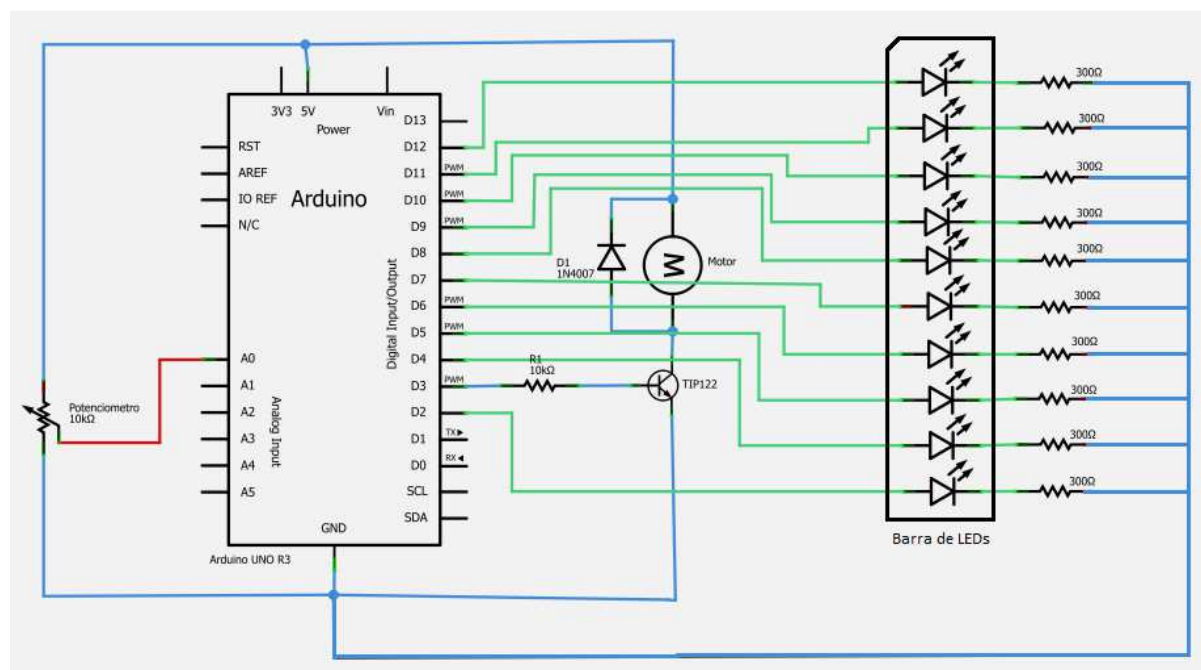
Se você fez uso do Kit Iniciante para Arduino da RoboCore, você já conhece esses conceitos. De toda forma, é sempre bom recordar:



A barra de LEDs é um componente muito usado em bargraphs em todo o mundo, em diversos equipamentos tanto comerciais quanto industriais. Ele serve justamente para, de uma forma fácil e inteligível, verificar como estão níveis de motores, de combustíveis, de volume, etc. Existem barras de LEDs de diversas cores. Seu encapsulamento e formato parecem muito um display de 7 segmentos, e de fato sua composição é praticamente a mesma, porém com uma diferença: a barra de LEDs não possui os terras ligados (os catodos não tem um ponto em comum). Desta forma, todos os pinos da barra de LEDs terão que ser usados.

Adicione a barra de LEDs ao seu circuito. Preste bastante atenção pois, além da barra ter polaridade (verifique a posição do chanfro no canto da barra), você precisará usar todos os pinos da mesma. De um lado, você deverá colocar os resistores para que a barra funcione e nenhum LED queime. Do outro lado você ligará no Arduino com os jumpers.

Sugerimos usar os próprios resistores para ligar cada catodo da barra para GND do Arduino, usando a barra horizontal da protoboard (que possui contato entre si). O circuito a ser montado é o seguinte:



Novamente, preste muita atenção e tenha calma ao montar o circuito. Ligando a barra de LEDs errado não irá danificar a mesma (desde que se use os resistores). Se após a ligação a barra não funcionar com o código, tire a barra da protoboard, vire 180° e ligue ao contrário (pode ser que você tenha errado a polaridade da mesma).

Feita a montagem, coloque no Arduino o seguinte código:

```
const int pwm = 3;
const int pot = A0;
int ValorLidoPot_Real = 0;
int ValorLidoPot_Mapeado = 0;
int BarraLed[] = {
  2, 4, 5, 6, 7, 8, 9, 10, 11, 12};

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

  for(int i = 0; i <= 9; i++){
    pinMode(BarraLed[i], OUTPUT);
  }
  for(int i = 0; i <= 9; i++){
    digitalWrite(BarraLed[i], LOW);
  }
}

void loop(){
  ValorLidoPot_Real = analogRead(pot);
  ValorLidoPot_Mapeado = map(ValorLidoPot_Real, 0, 1023, 0, 255);
  analogWrite(pwm, ValorLidoPot_Mapeado);
  Serial.println(ValorLidoPot_Mapeado);

  if(ValorLidoPot_Mapeado > 25){
    digitalWrite(BarraLed[0], HIGH);
    if(ValorLidoPot_Mapeado > 50){
      digitalWrite(BarraLed[1], HIGH);
      if(ValorLidoPot_Mapeado > 75){
        digitalWrite(BarraLed[2], HIGH);
        if(ValorLidoPot_Mapeado > 100){
          digitalWrite(BarraLed[3], HIGH);
          if(ValorLidoPot_Mapeado > 125){
            digitalWrite(BarraLed[4], HIGH);
            if(ValorLidoPot_Mapeado > 150){
              digitalWrite(BarraLed[5], HIGH);
              if(ValorLidoPot_Mapeado > 175){
                digitalWrite(BarraLed[6], HIGH);
                if(ValorLidoPot_Mapeado > 200){
                  digitalWrite(BarraLed[7], HIGH);
                  if(ValorLidoPot_Mapeado > 225){
                    digitalWrite(BarraLed[8], HIGH);
                    if(ValorLidoPot_Mapeado > 250){
                      digitalWrite(BarraLed[9], HIGH);
                      if(ValorLidoPot_Mapeado > 255){
                        digitalWrite(BarraLed[10], HIGH);
                      }
                    } else digitalWrite(BarraLed[10], LOW);
                  }
                } else digitalWrite(BarraLed[9], LOW);
              }
            } else digitalWrite(BarraLed[8], LOW);
          }
        } else digitalWrite(BarraLed[7], LOW);
      }
    } else digitalWrite(BarraLed[6], LOW);
  }
} else digitalWrite(BarraLed[5], LOW);
}
} else digitalWrite(BarraLed[4], LOW);
}
} else digitalWrite(BarraLed[3], LOW);
}
} else digitalWrite(BarraLed[2], LOW);
}
} else digitalWrite(BarraLed[1], LOW);
}
} else digitalWrite(BarraLed[0], LOW);
}
```

Sim, o código é grande, porém não é muito complexo. O que mudou do código anterior para este é que agora temos que declarar as 10 variáveis dos pinos digitais onde existem LEDs (da barra de LEDs). Na rotina de setup também temos que dizer que os 10 pinos são saídas. Veja que foi usada uma rotina *for* para isto. Isto é um artifício que serve para diminuir o código. Também foi usada uma rotina *for* para dizer que todos os LEDs devem começar apagados. Na parte do loop do programa existem diversas rotinas *if*, uma dentro da outra, que servem para acender os LEDs quando os valores lidos pelo potenciômetro forem maiores que determinados valores. Já que usamos um artifício para diminuir a parte do setup do programa, por que não usar algum artifício para diminuir a parte do loop? Da forma que está no loop funciona, porém é uma forma leiga de escrever o código. Ao ter experiência com programação, podemos mudar tudo aquilo para o seguinte código:

```
const int pwm = 3;
const int pot = A0;
int ValorLidoPot_Real = 0;
int ValorLidoPot_Mapeado = 0;
int BarraLed[] = {
  2, 4, 5, 6, 7, 8, 9, 10, 11, 12};

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

  for(int i = 0; i <= 9; i++){
    pinMode(BarraLed[i],OUTPUT);
  }
  for(int i = 0; i <= 9; i++){
    digitalWrite(BarraLed[i],LOW);
  }
}

void loop(){
  ValorLidoPot_Real = analogRead(pot);
  ValorLidoPot_Mapeado = map(ValorLidoPot_Real, 0, 1023, 0, 255);
  analogWrite(pwm, ValorLidoPot_Mapeado);
  Serial.println(ValorLidoPot_Mapeado);

  for(int i = 0; i <= 9; i++){
    if(ValorLidoPot_Mapeado > ((i+1)*25)){
      digitalWrite(BarraLed[i], HIGH);
    }
    else{
      digitalWrite(BarraLed[i], LOW);
    }
  }
}
```

Deixamos o entendimento deste código para o leitor. Apenas podemos advertir que ao invés de chegar a 255, desta maneira o motor chega apenas até 250. Você sabe dizer por quê?

- **Utilizando LEDs Infravermelhos**

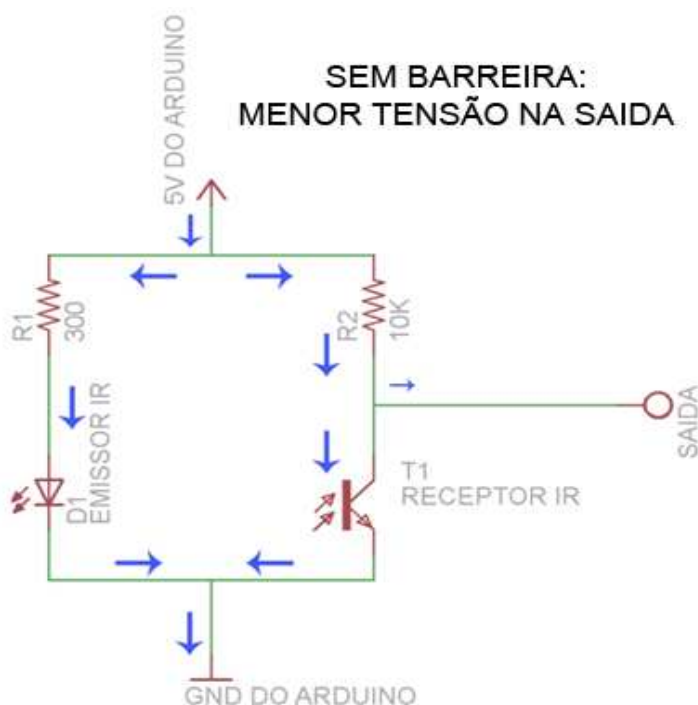
Componentes: 01x LED Receptor Infravermelho + 01x LED Emissor Infravermelho

Descrição: Aprenda a fazer barreiras invisíveis a olho nu.

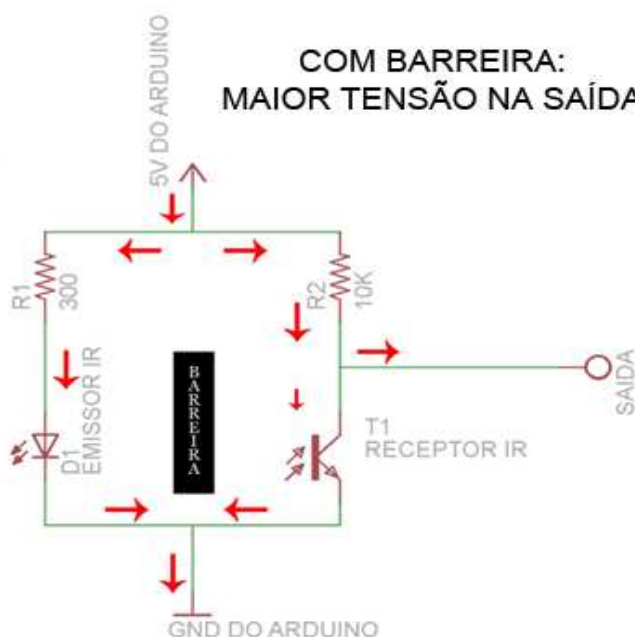
LEDs que emitem e recebem raios infravermelhos estão por toda parte. Com diferentes encapsulamentos, eles podem ser encontrados desde sensores de presença até controles remotos. Neste experimento, iremos entender como funcionam aquelas barreiras invisíveis a olho nu, que muitas vezes fazem a segurança de muros e portões de empresas, condomínios e prédios.

Os LEDs que iremos utilizar neste experimento parecem LEDs que foram achatados, por causa de seu formato diferenciado. Eles são quadrados, ao invés de redondos como aqueles que emitem luzes convencionais. Tenha sempre em mente que são LEDs, portanto aquele que é o **emissor** irá emitir uma luz que, por ser infravermelha, é invisível aos nossos olhos (algumas câmeras conseguem captar esta luz). Também pensando nisso é interessante lembrar que iremos precisar de um **resistor** junto ao LED emissor para que o mesmo não queime ao ser aceso.

O LED receptor é diferente. Ele na verdade é um fototransistor. Muito parecido a um transistor convencional, porém ao invés de termos um sinal vindo de um microcontrolador em sua base, temos uma luz infravermelha. Quando o raio infravermelho incidir sobre o LED receptor (fototransistor), haverá a passagem de tensão entre coletor e emissor do fototransistor. Isto fica muito mais fácil e inteligível vendo na imagem a seguir:



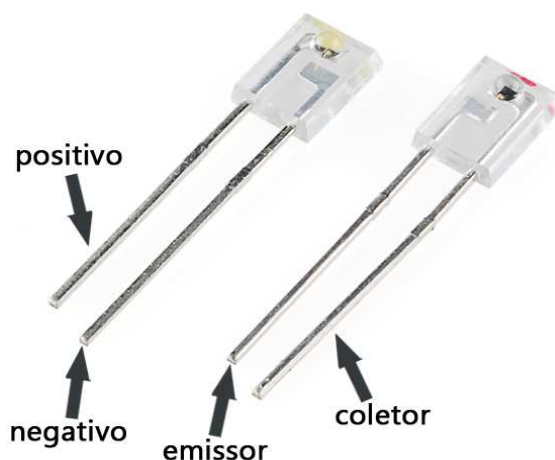
Veja que quando não há nada impedindo que o raio infravermelho do LED emissor chegue ao LED receptor, praticamente toda a tensão passa pelo coletor e emissor do transistor, e pouquíssima tensão vai para a saída. As coisas mudam bastante quando colocamos uma barreira entre o emissor e o receptor. Veja o esquema a seguir:



Veja que neste caso, como a luz infravermelha do LED emissor está "parando" na barreira, ela não consegue chegar até a base do fototransistor. Desta forma, a maior parte da tensão vai para a saída (que, fundamentalmente, pode ser a entrada analógica de seu Arduino). Portanto, analisando as duas imagens anteriores podemos presumir que, quando não tivermos barreira o valor lido em nossa porta analógica será **BAIXO** e quando houver barreira, teremos mais corrente chegando ao Arduino, por isso teremos um valor mais **ALTO**.

Chega de teoria, vamos para a prática. Monte o circuito anterior em seu Arduino. Para saber qual LED é o receptor e qual é o emissor, olhe na parte superior dos dois. Um deles tem um pontinho vermelho e o outro tem um pontinho amarelo. O que tem o pontinho **VERMELHO** é o **DETECTOR**. Já o que tem o pontinho **AMARELO** é o **EMISSOR** de luz infravermelha. Você, então, usará um resistor de 300 Ohms com o emissor, ou seja, o LED que tem o ponto amarelo, e um resistor de 10K com o receptor. Como qualquer LED, eles tem polaridade. Isto quer dizer que se você ligar ao contrário, o circuito não irá funcionar.

Veja a seguinte imagem para saber como ligar corretamente:



E não se esqueça do principal: ligue a **SAÍDA** no pino analógico **0** do Arduino.

Dica: Para ter resultados mais satisfatórios, procure deixar o emissor virado de frente para o receptor, a uma distância de no máximo 5cm.

Grave o seguinte código em seu Arduino:

```
const int Receptor = A0;
int ValorSensor = 0;

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

void loop() {
  ValorSensor = analogRead(Receptor);
  Serial.println(ValorSensor);
  delay(300);
}
```

Grave o código acima no Arduino e abra o monitor serial do programa do Arduino. Sem nenhuma barreira você deverá verificar valores baixos, uma vez que toda a tensão está praticamente passando pelo LED receptor, uma vez que a luz infravermelha está incidindo na base do fototransistor. Agora, use sua mão como barreira e interfira na passagem de luz entre LED emissor e receptor. Você deverá ver os valores no monitor serial aumentarem consideravelmente, isto porque o fototransistor não está mais conduzindo então praticamente toda a tensão vai para a entrada analógica do Arduino.

Agora você já sabe trabalhar com este tipo de LED, e pode fazer, por exemplo, um controle de acesso ou mesmo um aviso de segurança caso uma porta ou portão sejam abertos. Colocando os dois LEDs próximos um ao outro, um no batente da porta e o outro na porta em si, quando a porta for aberta você consegue detectar isto com o Arduino, e pode, conseqüentemente, fazer algo com isto, como por exemplo soar um alarme, tocar uma buzina, ou mesmo algo mais inteligente como acender a luz. Desta forma, a luz ficaria acesa sempre que a porta estivesse aberta.

Este é apenas um exemplo do que pode ser feito com este par de LEDs. Eles também podem ser usados na robótica como sensores de linha, por exemplo. Conhecendo o conceito, o projeto se torna mais fácil e é este o intuito deste material. Além disso você ainda pode usar estes LEDs como emissores e receptores de sinais de controle remoto. Veremos isto mais pra frente neste material.

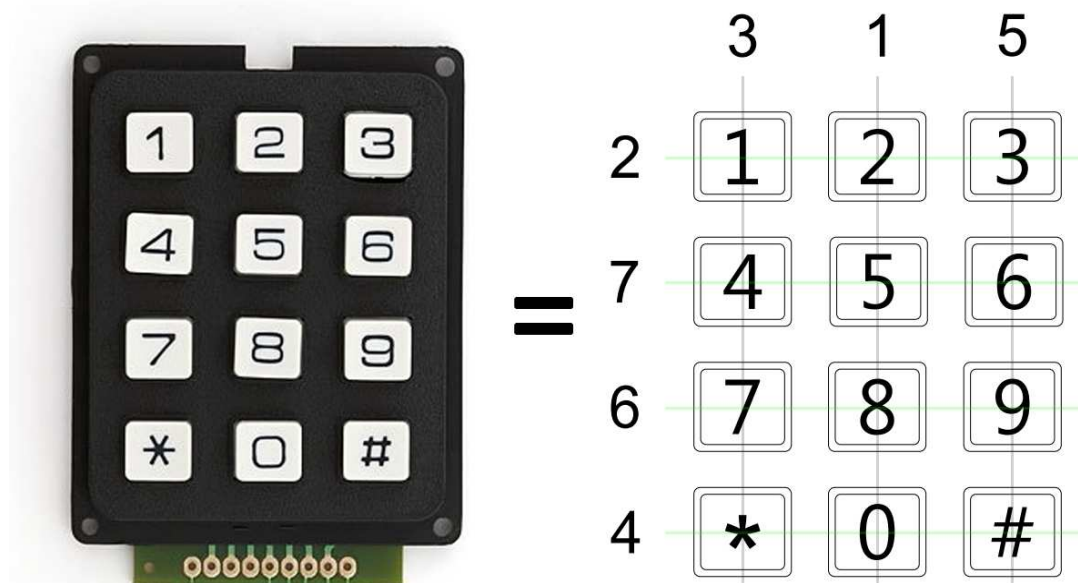
Apenas um detalhe: as lâmpadas que temos em casa, na escola, faculdade ou trabalho também emitem luz infravermelha. Isto quer dizer que elas podem interferir neste experimento. Caso você não esteja tendo sucesso em algum experimento, experimente cobrir os LEDs da luz externa e verifique se o sistema funciona de maneira mais satisfatória.

- **Teclado de 12 Botões**

Componentes: 01xTeclado com 12 botões + 01x Servo Motor

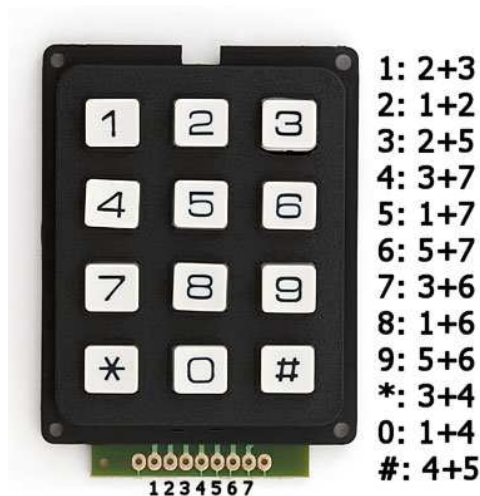
Descrição: Aprenda a usar um teclado com 12 botões, muito utilizado em telefones e em dispositivos eletrônicos onde você deve colocar uma senha para acessar um recinto.

Parece coisa de filme de agente secreto, mas não é! O controle de acesso colocando senhas numéricas está em toda parte e você, a partir de agora, vai poder fazer seu próprio sistema de controle de acesso via senha. O teclado com 12 botões que acompanha este kit nada mais é que uma matriz de contatos. Dê uma olhada na imagem a seguir:



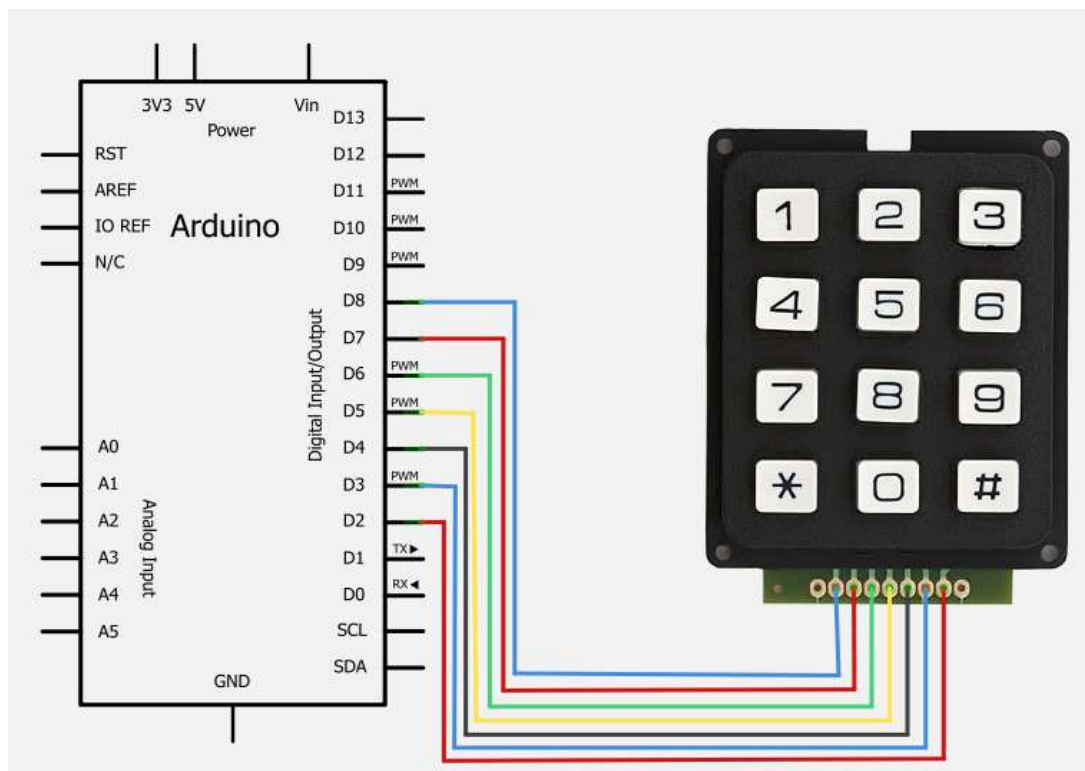
Esta imagem diz muito sobre o teclado. Com ela entendemos que, quando apertamos o número **5**, por exemplo, temos a junção entre o pino 1 e o pino 7. Já, quando apertamos o número **6** teremos a junção do pino 7 com o pino 5. Se você entendeu, tente fazer, mentalmente, o mapeamento de todos os botões. Isto será muito útil para entender a programação.

Para te ajudar, veja abaixo o mapeamento completo:



Mas como fazer a leitura do teclado no Arduino?

Temos que, de alguma forma, conseguir ler no Arduino dois pinos por vez para achar quem está pressionado. No Arduino temos portas de entrada analógicas e digitais. Neste caso, como somente queremos saber se existe ou não corrente passando por ali, iremos utilizar portas digitais. Ao todo, serão 7 portas digitais, uma para cada linha e coluna da matriz de contatos. Vamos entender daqui a pouco o que vamos fazer. Primeiro, monte o seguinte circuito:



E agora, grave o seguinte código em seu Arduino:

```
const int numLinhas = 4; //número de linhas do teclado
const int numColunas = 3; //número de colunas do teclado
const int debounce = 20; //tempo de pausa
const char teclado[numLinhas][numColunas] = { //definição do teclado
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}};

const int PinosLINHAS[numLinhas] = { 7, 2, 3, 5}; //pinos do Arduino que
correspondem às linhas do teclado
const int PinosCOLUNAS[numColunas] = { 6, 8, 4}; //pinos do Arduino que
correspondem às colunas do teclado
const char KNADA = 0; //constante que simboliza nenhuma tecla pressionada (deve ser
diferente de todos os valores em teclado[][] - o 0 desta variável não é o 0 do
teclado)
//-----
void setup(){
  Serial.begin(9600); //inicializa a Serial

  //configura as linhas como entradas com padrão em nível lógico alto
  for(int linha = 0; linha < numLinhas; linha++){
    pinMode(PinosLINHAS[linha], INPUT);
    digitalWrite(PinosLINHAS[linha], HIGH); //habilita pull-up
  }
  //configura as colunas como saídas com padrão em nível lógico alto
  for(int coluna = 0; coluna < numColunas; coluna++){
    pinMode(PinosCOLUNAS[coluna], OUTPUT);
    digitalWrite(PinosCOLUNAS[coluna], HIGH);
  }

  Serial.println("--- pressione uma tecla ---");
}
//-----

void loop(){
  char tecla = TeclaPressionada();
  if(tecla != KNADA){
    Serial.print("Tecla pressionada: ");
    Serial.println(tecla);
  }
}
//Determina qual tecla foi pressionada
char TeclaPressionada(){
  char tecla = KNADA; //atribui o valor padrão de retorno (nenhuma tecla
pressionada)
  boolean achou = false; //inicialmente considera que nenhuma tecla foi pressionada
  for(int coluna = 0; coluna < numColunas; coluna++){
    digitalWrite(PinosCOLUNAS[coluna], LOW); //muda o estado do pino
    //varre as linhas procurando por uma tecla pressionada
    for(int linha = 0; linha < numLinhas; linha++){
      //lê linha pela primeira vez
      if(digitalRead(PinosLINHAS[linha]) == LOW){ //tecla está pressionada
        delay(debounce); //insere pausa para descartar ruídos
      }
      //lê linha pela segunda vez
      if(digitalRead(PinosLINHAS[linha]) == LOW){ //tecla continua pressionada,
portanto não é um ruído
        while(digitalRead(PinosLINHAS[linha]) != HIGH); //espera soltar o tecla
para retornar
        tecla = teclado[linha][coluna]; //determina qual foi a tecla pressionada de
acordo com o teclado definido
        achou = true; //uma tecla foi pressionada
        break; //sai da varredura das linhas
      }
    }
    digitalWrite(PinosCOLUNAS[coluna], HIGH); //retorna o pino ao seu estado
inicial
  }
}
//continua na próxima página
```

```
//continuação:

//se uma tecla foi pressionada, sai do laço for
if(achou){
    break;
}

return tecla; //retorna a tecla pressionada (KNADA se nenhuma foi encontrada)
}
```

Por ser um código um pouco mais complexo, ele está completamente comentado. Para fazer o código funcionar perfeitamente é aconselhável deixar a IDE do Arduino maximizada em seu computador, assim as linhas de comentário ficarão em sua posição correta e o programa não irá pensar que os comentários são linhas de código.

Este código mantém todos os pinos que estão dispostos horizontalmente como entrada e também setamos estes pinos em nível lógico alto. Isto parece ser estranho, já que quando setamos o pino como entrada esperamos que venha um nível lógico alto ou baixo, e não já deixamos setados. Quando setamos um pino como entrada e também setamos seu nível lógico como alto, setamos o pull-up interno do Arduino, o que irá permitir um sensoramento se o pino for usado (quando pressionarmos algum botão no teclado). Depois, setamos os pinos de coluna como saídas e também fazemos seu nível lógico ficar em alto.

A lógica da coisa é ficar chaveando os pinos das linhas e das colunas, em uma determinada ordem, colocando um pino de cada vez das colunas em nível lógico baixo e ir monitorando se este nível lógico baixo entra em algum pino de linha. Como os pinos de linha estão em alto, quando entrar o nível baixo, o Arduino irá saber através do comando **digitalRead(PinosLINHAS)**. Se entrar, quer dizer que alguma tecla foi apertada. Este chaveamento ocorre de uma forma muito rápida, e o Arduino tem capacidade para fazer isso em tal velocidade que a qualquer momento você pode apertar uma tecla e ela será detectada. Apenas lembrando que o código contempla apenas uma tecla apertada de cada vez.

O nível de complexidade deste código é alto. Se você conseguir entendê-lo completamente pode ter certeza que está no caminho certo nesta jornada na tecnologia Arduino.

Vamos agora utilizar isto para alguma coisa. Continue com o mesmo circuito montado, e adicione um Servo Motor no mesmo. Lembre-se, o pino mais escuro do servo vai no **GND** da placa, o pino logo ao lado do mais escuro vai no **5V** e o último pino vai em um pino com PWM, neste caso usaremos o **pino digital 9**. Neste experimento, teremos diversas posições do servo salvas no código, e chamaremos cada posição de maneira rápida utilizando os botões do teclado.

```
#include <Servo.h>
Servo servo;

const int numLinhas = 4;
const int numColunas = 3;
const int debounce = 20;
const char teclado[numLinhas][numColunas] = {
    {'1','2','3'},
    {'4','5','6'},
    {'7','8','9'},
    {'*','0','#'}};

const int PinosLINHAS[numLinhas] = { 7, 2, 3, 5};
const int PinosCOLUNAS[numColunas] = { 6, 8, 4};
const char KNADA = 0;

//continua na proxima pagina
```

```
void setup(){
  servo.attach(9);
  Serial.begin(9600);
  for(int linha = 0; linha < numLinhas; linha++){
    pinMode(PinosLINHAS[linha], INPUT);
    digitalWrite(PinosLINHAS[linha], HIGH); //habilita pull-up
  }

  for(int coluna = 0; coluna < numColunas; coluna++){
    pinMode(PinosCOLUNAS[coluna], OUTPUT);
    digitalWrite(PinosCOLUNAS[coluna], HIGH);
  }
  Serial.println("--- pressione uma tecla ---");
}

void loop(){
  char tecla = TeclaPressionada();
  if(tecla != kNADA){
    Serial.print("Posicao do eixo do servo: ");
    if(tecla == '1'){
      servo.write(10);
      Serial.println("10 graus");
    }
    else if(tecla == '2'){
      servo.write(20);
      Serial.println("20 graus");
    }
    else if(tecla == '3'){
      servo.write(30);
      Serial.println("30 graus");
    }
    else if(tecla == '4'){
      servo.write(40);
      Serial.println("40 graus");
    }
    else if(tecla == '5'){
      servo.write(50);
      Serial.println("50 graus");
    }
    else if(tecla == '6'){
      servo.write(60);
      Serial.println("60 graus");
    }
    else if(tecla == '7'){
      servo.write(70);
      Serial.println("70 graus");
    }
    else if(tecla == '8'){
      servo.write(80);
      Serial.println("80 graus");
    }
    else if(tecla == '9'){
      servo.write(90);
      Serial.println("90 graus");
    }
    else if(tecla == '*'){
      servo.write(150);
      Serial.println("150 graus");
    }
    else if(tecla == '0'){
      servo.write(0);
      Serial.println("0 graus");
    }
    else if(tecla == '#'){
      servo.write(180);
      Serial.println("180 graus");
    }
  }
}

//continua na proxima pagina
```

```
char TeclaPressionada(){
  char tecla = KNADA;
  boolean achou = false;
  for(int coluna = 0; coluna < numColunas; coluna++){
    digitalWrite(PinosCOLUNAS[coluna], LOW);
    for(int linha = 0; linha < numLinhas; linha++){
      if(digitalRead(PinosLINHAS[linha]) == LOW){
        delay(debounce);
      }

      if(digitalRead(PinosLINHAS[linha]) == LOW){
        while(digitalRead(PinosLINHAS[linha]) != HIGH);
        tecla = teclado[linha][coluna];
        achou = true;
        break;
      }
    }
    digitalWrite(PinosCOLUNAS[coluna], HIGH);
    if(achou){
      break;
    }
  }
  return tecla;
} //fim do código
```

A única mudança entre o código acima e o anterior é que agora fazemos algo mais do que apenas mostrar no monitor serial qual tecla foi pressionada. Com rotinas **IF** é possível executar quaisquer funções apenas apertando cada tecla.

Para fazer um sistema de senhas, por exemplo, seria necessário gerar uma variável no início do programa que contém a senha exata que daria acesso, por exemplo, a uma porta. Então, teria que ser feito um buffer da mensagem recebida e este buffer acabaria quando fosse apertada, por exemplo, a tecla #, finalizando a senha (seria o ENTER). Para tornar as coisas mais fáceis, existe uma biblioteca pronta para este sistema de senhas. Se o usuário deste kit tiver interesse, pode entrar no seguinte site: <http://bildr.org/2011/05/arduino-keypad/> e verificar o projeto para senhas.

- **Circuito Integrado 555**

Componentes: 01x Circuito Integrado 555

Descrição: Veja algumas, das várias de possibilidades de projetos, que este pequeno circuito integrado pode fazer.

Este, talvez, é o circuito integrado (CI) mais versátil e um dos mais usados no mundo. O que o torna tão versátil são os diversos modos de funcionamento. Em seu datasheet existe uma descrição completa, inclusive com exemplos, do que é e para que serve este componente. De forma breve, este CI pode trabalhar na operação **Monoestável**, **Astável**, **Modulador por Largura de Pulso** e **Rampa Linear**. Cada um dos modos possui sua determinada função.

Modo monoestável: o CI trabalha com a função chamada "one-shot", ou seja, um tiro. Isto quer dizer que, quando houver a entrada de um sinal em uma determinada porta do CI, em sua saída irá aparecer uma onda com uma largura fixa, sem variação. Cada vez que houver um pulso de gatilho, aparecerá exatamente a mesma forma de onda na saída. O bacana deste modo é que, mesmo se entrar outro sinal de gatilho, o CI não fará nada até que a resposta ao anterior termine (não havendo possibilidade de "encavalar" dois comandos). A duração do pulso de saída neste modo é dado pela equação $t = 1,1 \times R_1 \times C_1$, onde R_1 e C_1 são um resistor e um capacitor, respectivamente, na ligação pertinente a este caso (vide página 6 do datasheet que se encontra na página deste produto no site da RoboCore).

Modo Astável: um dos modos mais usados, com ele é possível gerar frequências para serem usadas como *clocks* de outros circuitos integrados, como microcontroladores, ou mesmo gerar notas musicais a partir de diferentes frequências. A fórmula para se determinar a frequência de operação na saída do 555 é: $f = \frac{1,44}{(R_1 + 2R_2) \times C_1}$, onde as variáveis da equação são componentes e suas posições no circuito podem ser vistas na página 7 do datasheet.

Modo Modulador de Largura de Pulso: Neste modo, o circuito integrado irá gerar um sinal de saída do tipo PWM, muito parecido com os sinais PWM que o Arduino possui.

Modo Rampa Linear: este modo é muito parecido com o modo monoestável, porém a forma de onda na saída do circuito neste caso será uma rampa linear.

Com este circuito integrado é possível fazer osciladores, geradores de onda, divisores de frequência, pianos de brinquedo, sirenes parecidas com as dos carros de polícia norte-americanos, detectores de metais, sinalizadores com duas lâmpadas (comum em entradas/saídas de veículos), dimmer para LEDs, controlador de motores via PWM, gerador de sinal PWM, aumentador de tensão com corrente não superior a 50mA, gerador de tensão negativa, amplificador, detector de luz, detector de penumbra, tacômetro para carros, medidor de frequência, testador de servo-motor, testador de transistor, testador de diodo-zener, testador de continuidade, gerador de código morse, controlador automático de luzes de semáforos, cubo de LED, além de muitos outros. Para montar a maioria dos projetos é necessário diversos valores de resistores e capacitores. Como em nosso kit temos apenas alguns valores deste componentes, disponíveis, iremos nos limitar a alguns experimentos. O que queremos mostrar aqui é que, utilizando um circuito integrado 555 muitas vezes é mais barato do que utilizar uma placa Arduino ou qualquer outro microcontrolador, e acaba ficando uma solução mais barata para alguns projetos.

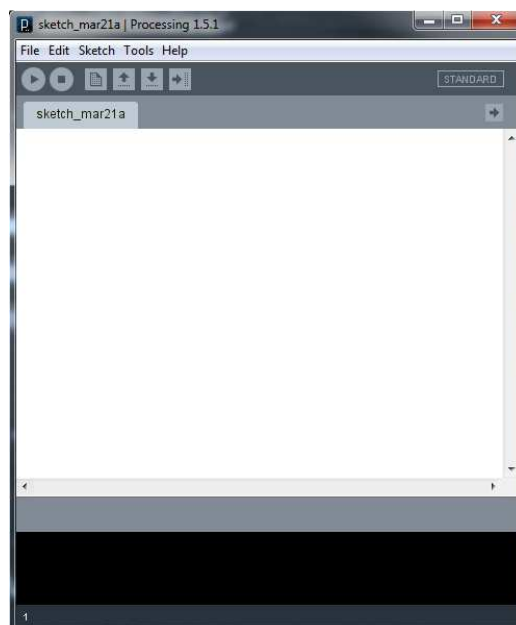
Mas, se para alguns projetos nós podemos escolher entre usar um Arduino OU usar um 555, o que podemos fazer com os dois dispositivos?

Todos os projetos usando o 555 tem uma coisa em comum: na saída do CI existe uma forma de onda (seja quadrada, rampa, etc). Portanto, o que podemos fazer com um Arduino é verificar se o formato da onda gerada pelo 555 está conforme o esperado. Uma maneira de fazer isso é colocar a saída do 555 em uma entrada analógica do Arduino e ler os valores pelo monitor serial, conforme fazemos em diversos experimentos no Kit Iniciante para Arduino da RoboCore. Porém, apenas olhando os valores numéricos não seria muito fácil e inteligível verificar se a forma de onda gerada é uma quadrada, retangular, rampa ou outra qualquer. Desta forma, teríamos que pegar os valores

obtidos no monitor serial e passar para um programa que gere gráficos, como o Excel da Microsoft®. O único problema é que desta forma nós não poderíamos ver em tempo real o formato da onda, o que tornaria inviável o processo. Desta forma, precisaríamos ver em tempo real o que o Arduino está recebendo de uma maneira gráfica visual e nada melhor para fazer isso do que o programa **PROCESSING**.



O software Processing nasceu antes mesmo da plataforma Arduino eclodir para o mundo. Trata-se de um programa muito parecido com a IDE do Arduino, porém é voltada para parte gráfica visual. Através de código e programação, é possível gerar belas imagens e interagir com elas, seja através de um teclado, mouse ou outro periférico. Um destes periféricos são as placas Arduino. Com o **Processing**, a ponte entre hardware Arduino e imagens gráficas visuais no computador foi tornada mais fácil. O **Processing** é uma ferramenta muito poderosa, e para ilustrar o que ela pode fazer lhe convidamos a copiar a pasta **Processing** presente no CD que acompanha este kit **para sua pasta raiz** (na maioria dos casos C: se for Windows ou instalar o App se for Mac). Sua interface é muito parecida com a IDE do Arduino, conforme podemos ver na seguinte imagem:



Ao passar o ponteiro do mouse sobre os principais botões da tela do programa, você verá que existem alguns muito parecidos com o da IDE do Arduino, como o botão de novo arquivo (**New**), abrir arquivo (**Open**), salvar arquivo (**Save**) e um botão para exportar o aplicativo (**Export Applet**) - este último não existe na IDE do Arduino. Antes destes botões existem outros dois. Um deles é o **Run** (rodar) e o **Stop** (parar). Veja que neste programa não temos um botão para compilar o código e para passar para a placa Arduino, pois ele não faz gravação de códigos na placa Arduino, ele apenas gera uma interface gráfica. Quando você escrever um código terá que apertar o botão **Run** para fazê-lo rodar e aparecer na tela. Para parar o programa que está rodando, basta clicar no **Stop**.

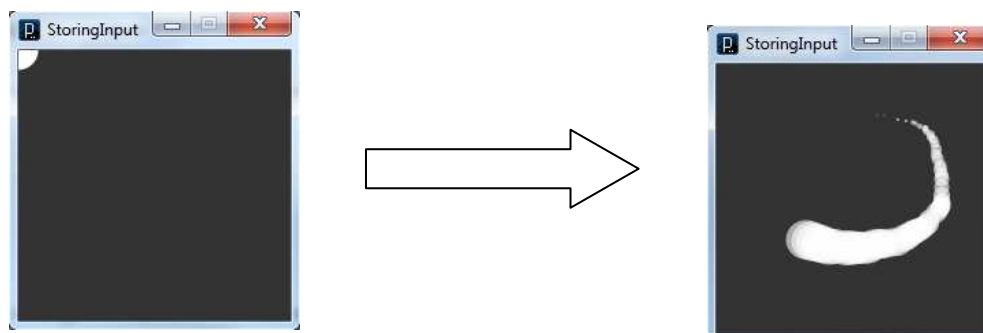
Vamos começar o estudo deste programa fazendo algo simples, abrindo um exemplo pronto para vermos o que este programa é capaz.

Clique em **File > Examples... > Basics > Input > StoreInput**

Após o código ser carregado na tela, clique no botão **RUN**.

Após clicar do botão para rodar o programa, deve abrir uma nova tela, pequena, cinza com um ponto branco no canto superior esquerdo. Coloque o mouse sobre a tela e fique mexendo o

ponteiro do mouse dentro desta nova tela. Você deve ver um efeito gráfico interessante, como o que é mostrado nas figuras a seguir:



Este efeito foi feito puramente com códigos, inclusive o tamanho da tela, cor de fundo, cor da bola branca, etc. Dê uma olhada agora no código que o exemplo possui. Ele parece muito com códigos feitos para Arduino, porém são códigos para **Processing**. A sintaxe é muito parecida, primeiramente são declaradas variáveis (neste caso uma do tipo **int** e duas do tipo **float**) e então são feitos os códigos que serão rodados apenas uma vez no programa, que estão dentro da rotina **setup()**. Assim como no Arduino, ali são colocadas as configurações do programa. Enquanto no Arduino setamos quem serão as portas de entrada e de saída, por exemplo, no **Processing** este é o espaço para informar o tamanho da tela gráfica, a cor de fundo, estilos, etc. Por fim, ao invés de termos um **void loop()** como no programa do Arduino, temos um **void draw()**, nada mais justo pois aqui queremos que o programa desenhe alguma coisa na tela (mesmo ele ficando em **loop** também).

Faça agora o seguinte teste, mude os valores em **size(200, 200)** para **size(800,600)**. Clique no **Run** e veja que a nova tela será maior. Este valor está em pixels.

Vamos agora fazer algo com Arduino e **Processing**, uma interface bastante simples a princípio. Nosso objetivo neste momento é acender o LED da placa Arduino usando um botão na tela do computador. Para fazer isso, primeiro temos que colocar um código no Arduino que entenda que ao receber um comando deve acender seu LED. Grave então em seu Arduino o seguinte código:

```
const int ledPin = 13;
int incomingByte;

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

void loop() {
  if (Serial.available() > 0) {
    incomingByte = Serial.read();
    if (incomingByte == 'L') {
      digitalWrite(ledPin, HIGH);
    }
    if (incomingByte == 'D') {
      digitalWrite(ledPin, LOW);
    }
  }
}
```

Após gravar o código acima no Arduino, abra o monitor serial do próprio programa do Arduino e envie uma letra L (de ligar) pela serial. Você deverá ver o LED da placa Arduino acender. Se enviar uma letra D (de desligar), o LED deverá apagar. Feito isso, vamos para a parte do **Processing**.

Crie um novo arquivo no **Processing** e coloque o seguinte código na tela:

```
//codigo para o Processing:
import processing.serial.*;
int value = 0;
Serial port;

void setup() {
    println(Serial.list());
    port = new Serial(this, Serial.list()[0], 9600);
}

void draw() {
    rect(25, 25, 50, 50);
    fill(value);
}

void mouseClicked() {
    if (value == 0) {
        value = 255;
        port.write('L');
    }
    else {
        value = 0;
        port.write('D');
    }
}
```

Veja que o código é pequeno e, aparentemente simples. Na primeira linha de código falamos para o **Processing** que iremos utilizar comunicação serial. Declaramos a variável *value*, que irá receber um valor de 0 ou 255. Estes valores representam as cores preta e branca, respectivamente, para o comando *fill()* presente no bloco **void draw()** do programa. Ainda no início do código, declaramos que haverá uma porta serial que deverá ser aberta para comunicação.

No **setup** do programa é feito uma leitura de todas as portas seriais encontradas pelo computador e o programa sozinho consegue encontrar a porta do Arduino e fazer a devida conexão (desde que a porta COM do Arduino seja a primeira na lista de portas COM do computador). Estes comandos nos poupam algum tempo de programação por fazer tudo de maneira automática.

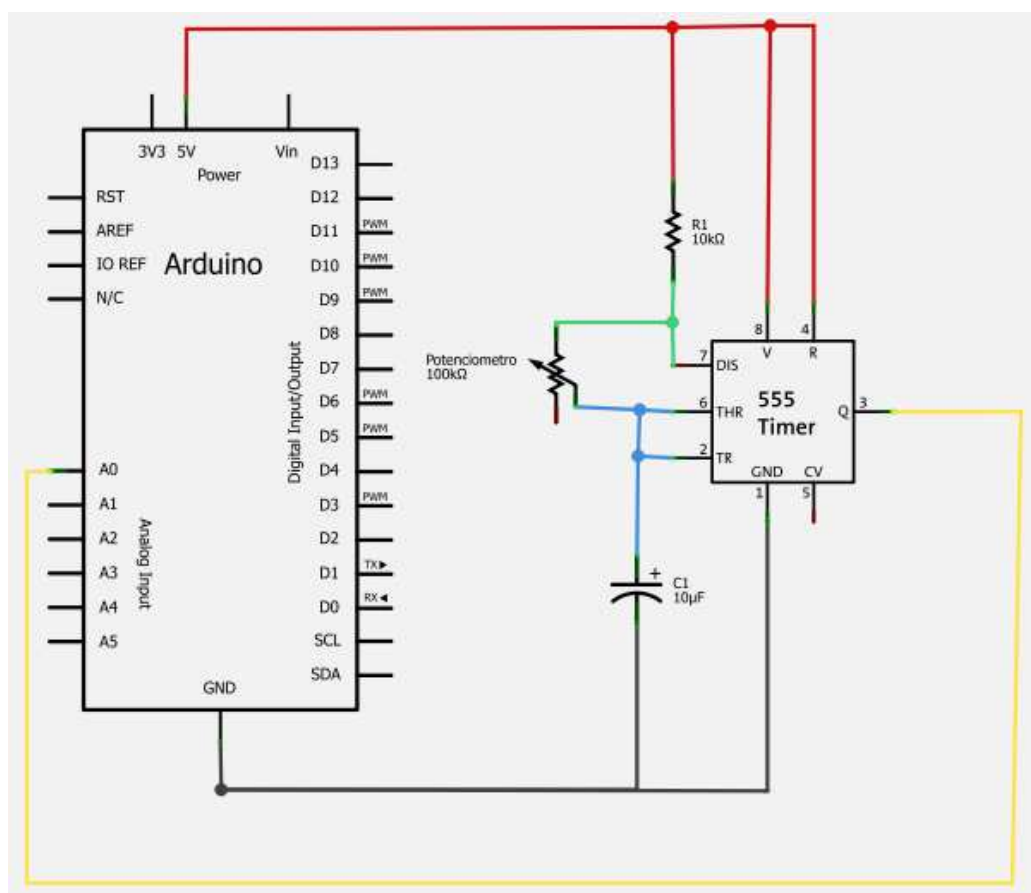
Seguindo o programa, chegamos ao bloco **void draw()**. Primeiramente, usamos o comando **rect(25, 25, 50, 50)**. Este comando serve para criar um quadrado em na tela gerada pelo programa. Os dois primeiros número (**25, 25**) servem para dizer a localização nas coordenadas x e y de onde o retângulo irá começar a ser desenhado. Já os números seguintes (**50, 50**) servem para determinar o tamanho, em pixels, do quadrado a ser desenhado. O próximo comando é o **fill()**, que pinta o quadrado desenhado de determinada cor. Como no início do programa dizemos que a variável **value** possui o valor **0**, o quadrado começará preto. Se mudarmos aquele valor para **255**, ele começará branco. O fato do quadrado estar preto, para nós, neste programa, irá mostrar que o LED do pino 13 está apagado. O fato do quadrado estar branco, vai indicar que o LED está aceso.

Diferente de programas feitos no Arduino, em **Processing** você não precisa necessariamente criar funções para verificar se houve algum evento, como o clique de um mouse ou uma tecla pressionada. Basta colocar a função já pronta para que o programa fique sempre verificando se houve um evento. Neste caso, colocamos a função **void mouseClicked()**, que ficará verificando se houve ou não o clique de um botão do mouse (qualquer um dos botões). Caso haja o clique, ele irá verificar qual o estado do LED, ou seja, se está aceso ou apagado. Ele faz esta checagem verificando o último valor salvo na variável **value** (que é a variável mais importante deste programa). Se a variável **value** for 0, quer dizer que nosso último comando foi de desligar a lâmpada, portanto temos que ligá-la. Desta forma, tornamos o valor do **value** 255 e jogamos pela porta serial uma letra **L**. Esta letra será recebida pelo Arduino através do cabo USB e, como temos um programa no Arduino que entende esta letra e faz determinada função, o LED será ligado. De uma forma semelhante, o LED será apagado caso cliquemos novamente no botão na tela do computador.

Existem diversos parâmetros que podemos utilizar do mouse, como qual botão do mouse foi apertado, posição x e y do ponteiro, e por aí vai. De forma parecida, podemos utilizar parâmetros vindos do teclado. Para projetos com gravação de dados, podemos gerar a partir do **Processing** arquivos de texto com dados vindos de sensores ligados ao Arduino, por exemplo, e muito mais. O **Processing** é uma ferramenta muito poderosa e abre todo um novo leque de possibilidades para quem o utiliza.

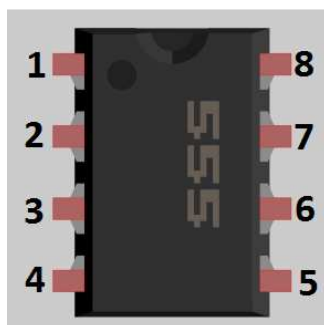
Agora que estamos um pouco mais familiarizados com este programa, vamos voltar ao experimento do circuito integrado 555. Iremos fazer no **Processing** uma espécie de osciloscópio para verificar formas de onda. Infelizmente não teremos como fazer medições com as curvas levantadas com o **Processing**, assim como poderíamos fazer em um osciloscópio real, porém ele nos "quebrará um galho" para vermos se a forma de onda esperada é a que estamos tendo na saída do 555. Lembre-se que para fazer medidas com precisão dos dados é sempre interessante introduzir o sinal da saída do 555 em um osciloscópio analógico ou digital. A placa Arduino não tem velocidade de processamento suficiente para trabalhar como um osciloscópio real.

Para começar, vamos montar o seguinte circuito em uma protoboard, usando o 555 e mais alguns componentes:



Componentes usados: 01x Potenciômetro de **100kΩ**, 01x Resistor de 10kΩ, 01x Capacitor de 10µF, 01x Circuito Integrado 555 e Arduino.

CUIDADO: Existe um lado certo para usar o circuito integrado 555. Ligando de forma errada você irá **queimar** o CI e deixá-lo inutilizável. Veja abaixo qual a posição correta:



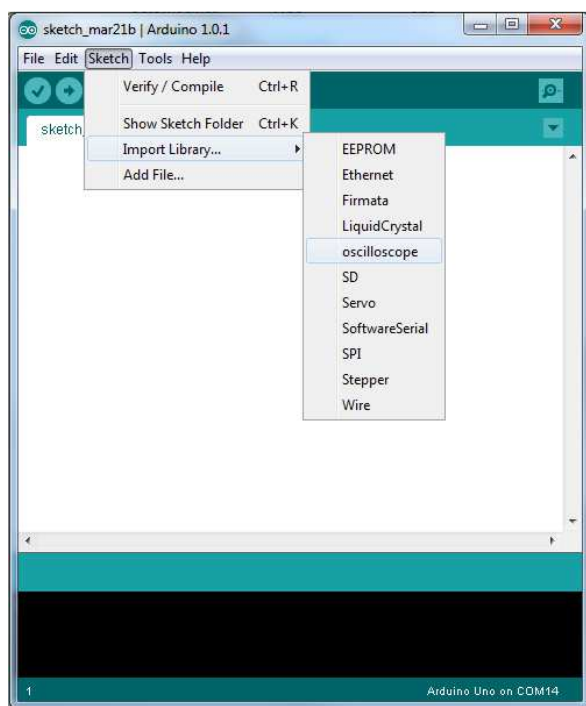
Dica: para saber qual o pino número 1, olhe para o chip e procure uma marcação em formato de uma pequena circunferência. O pino 1 é o mais próximo a ela.

Lembre-se também que o capacitor eletrolítico possui **polaridade**, portanto verifique o sinal de negativo no capacitor e ligue-o no GND.

Após a montagem do circuito acima, você precisará colocar no Arduino o código para enviar os valores pela porta serial. Antes de fazer isto, você precisa copiar a biblioteca que usaremos no código para sua pasta de bibliotecas. A biblioteca a ser copiada encontra-se dentro do CD que acompanha este kit, na pasta **Bibliotecas**. Copie a pasta chamada **oscilloscope** para a pasta **libraries** presente na pasta do Arduino (o ambiente de desenvolvimento do Arduino deve estar, preferencialmente, em sua pasta raiz C: - no Mac, clique com o botão direito do mouse e depois em *show packages* para visualizar a pasta *libraries*). Caso o programa Arduino esteja em sua pasta raiz no Windows, o caminho até a biblioteca deve ser o seguinte:

C:\Arduino\arduino-1.x\libraries\oscilloscope

Após copiar a pasta da biblioteca, reinicie o programa do Arduino. Para verificar se a biblioteca foi corretamente inserida na IDE do Arduino basta fazer o seguinte teste: entre na IDE, vá em **Sketch > Import Library...** lá deve aparecer a opção de clicar em **oscilloscope**. Não é necessário clicar, apenas veja se está lá para verificar se o procedimento foi feito com sucesso. Caso não esteja como na figura a seguir, você deverá refazer o procedimento:



Se está tudo nos conformes, grave o seguinte código no Arduino:

```
#include "oscilloscope.h"

#define ANALOG_IN 0

void setup() {
  Serial.begin(115200);
}

void loop() {
  int val = analogRead(ANALOG_IN);
  writeOscilloscope(val);
  delay(50);
}
```

O código é bastante simples:

Na primeira linha a biblioteca é importada e então fazemos a definição que o pino analógico 0 será chamado de **ANALOG_IN**, ou seja, entrada analógica.

No bloco **setup()** informamos que a comunicação se dará na taxa de 115200bps, a mais rápida que o Arduino suporta.

Finalmente no bloco de **loop()**, gravamos em uma variável chamada **val** (a qual declaramos na mesma hora, por isso o **int** antes do nome da variável) o valor lido na porta analógica de entrada, no caso analógica 0. Após isso, simplesmente escrevemos **writeOscilloscope(val)**, onde jogamos o valor lido para o computador através do cabo USB. Este comando não é comum do Arduino, nós apenas podemos usá-lo, pois ele está presente dentro da biblioteca **oscilloscope.h** que declaramos no início do código. Isto faz com que tudo seja feito automaticamente pela biblioteca.

Com o código gravado na placa, abra um novo arquivo no **Processing** e coloque o seguinte código nele:

```
/*
 * Código para visualizar portas analógicas no computador, como um osciloscópio.
 * Este código pode ser usado/distribuído através dos termos da GNU
 * Saiba mais em: <http://www.gnu.org/licenses/>.
 */
import processing.serial.*;
PFont f;
Serial port; // Cria objeto para a classe Serial
int val; // Variável para guardar dados da vindos da porta serial
int[] values;

void setup()
{
  size(660, 480); //cria uma tela de 640 x 480
  //o comando abaixo abre a porta que a placa está conectada e usa a mesma velocidade (115200 bps)
  port = new Serial(this, Serial.list()[0], 115200);
  values = new int[width];
  smooth();
  f = createFont("Arial",16,true); // Arial, 16 point, anti-aliasing on
}

int getY(int val) {
  return (int)(val / 1023.0f * height) - 1;
}

//continua na próxima página
```

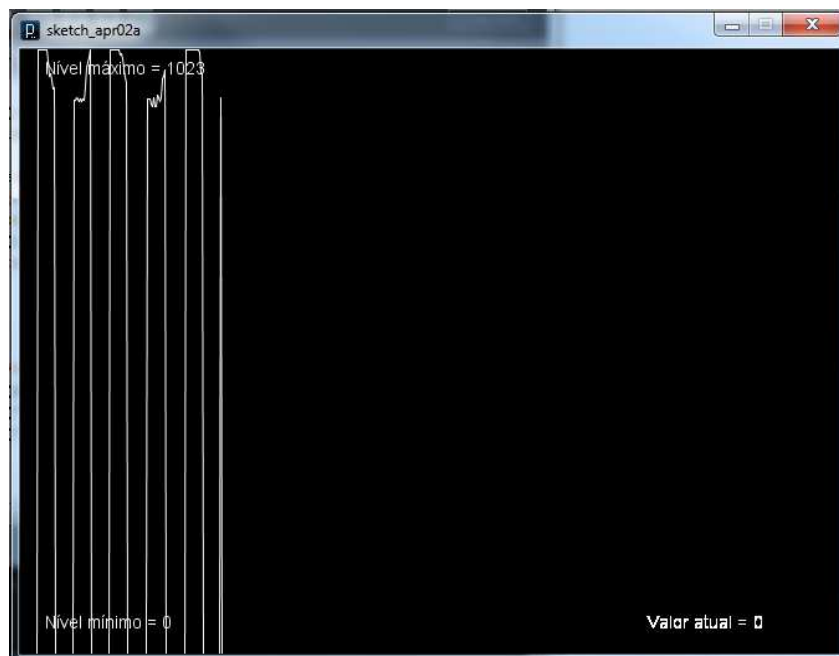
```
//continuacao:

void draw()
{
  while (port.available() >= 3) {
    if (port.read() == 0xff) {
      val = (port.read() << 8) | (port.read());
    }
  }
  for (int i=0; i<width-1; i++)
    values[i] = values[i+1];

  values[width-1] = val;
  background(0);
  stroke(255);
  for (int x=1; x<width; x++) {
    line(width-x, height-1-getY(values[x-1]),
         width-1-x, height-1-getY(values[x]));
    text("Valor atual = ", 505, 460);
    text(val, 590,460);
  }
  textFont(f,14); //aqui escrevemos as frases de valores mínimo e máximo
  fill(255);
  text("Nível máximo = 1023",20,20);
  text("Nível mínimo = 0",20,460);
}
```

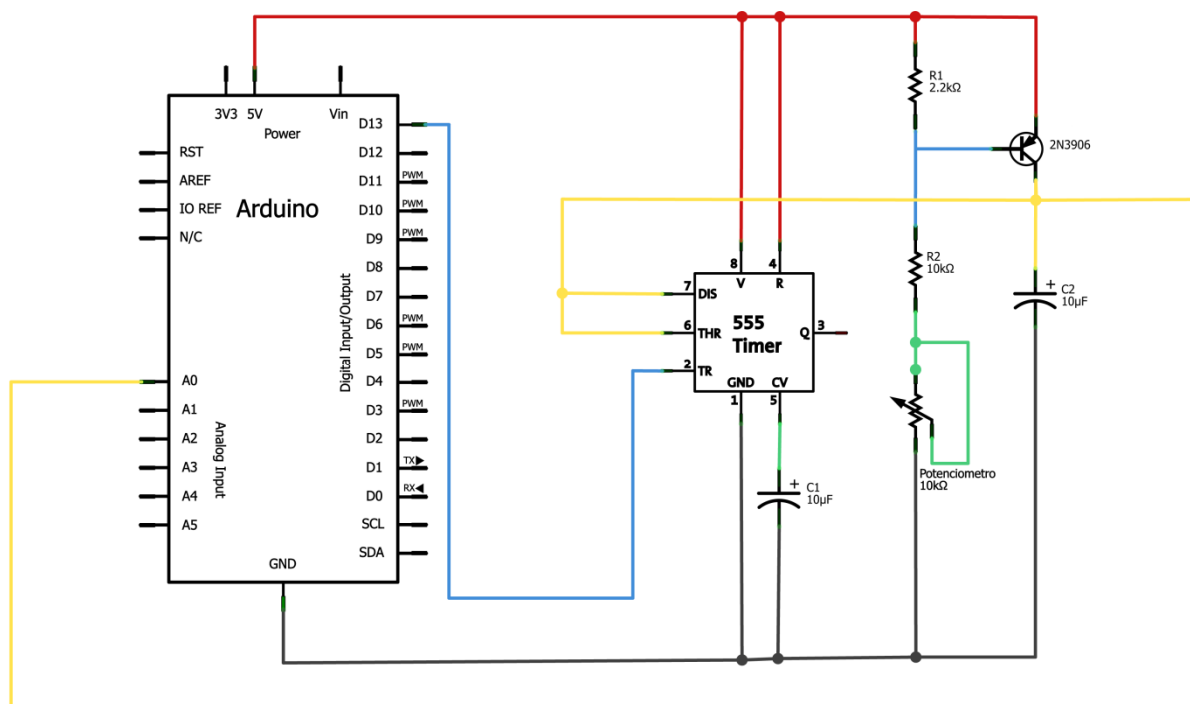
O projeto acima está comentado, porém vale dizer que o que ele faz é receber os valores vindos da porta serial e desenhar um ponto na tela. Os valores recebidos vão de 0 a 1023, pois estamos usando uma placa Arduino que possui um conversor analógico digital de 10 bits. Portanto, se recebermos o valor 0 ele pinta um ponto na parte de baixo da tela. Se recebermos 1023 ele pinta um ponto na parte de cima da tela, e tudo que estiver entre estes valores será plotado, formando assim uma forma de onda.

Clique no botão de rodar (RUN) no **Processing**, e devemos ver o seguinte:

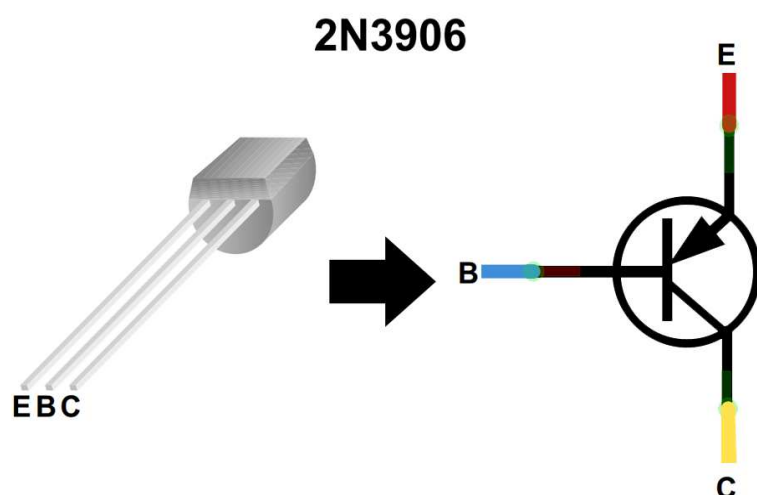


Veja que a partir do esboço da curva vista, podemos ver que temos de fato uma onda retangular na saída de nosso 555. Estamos trabalhando com ele no modo astável então era o esperado. Ainda sobre o modo astável, vale aquela equação que vimos anteriormente em relação à frequência de saída.

Vamos agora alterar o circuito que temos, para trabalhar com o 555 no modo de rampa linear. Desligue a placa Arduino do cabo USB, para desenergizar o sistema e altere o circuito para o seguinte (é aconselhável desmontar toda a montagem e refazê-la para evitar erros):



O circuito acima provavelmente é o mais difícil desta apostila. Veja que o potenciômetro agora é de 10kΩ. Você deve ter muita atenção ao montá-lo. Veja que existem várias ligações juntas, como a porta Analógica 0 do Arduino, que está ligada ao pino 6 e 7 do 555 e todos estão ligados no **coletor** do transistor e também no polo positivo do capacitor de 10uF. De novidade no circuito existe apenas um transistor do tipo PNP, diferente do TIP122 que usamos no experimento do motor que era NPN. O princípio de funcionamento é um pouco diferente. Neste experimento estamos usando o PNP de modelo 2N3906, um dos transistores BJT tipo PNP mais usados para baixas correntes. Veja que ele tem posição certa para conexão, para usá-lo, baseie-se na seguinte imagem:



Dica: tome o tempo que precisar para montar o circuito, não tenha pressa. Trata-se realmente de um circuito complexo e se você não prestar muita atenção ele não irá funcionar.

A pressa é inimiga da eletrônica!

Após montar o circuito em questão, grave no Arduino o seguinte código:

```
#include "oscilloscope.h"

#define ANALOG_IN 0
const int ledPin = 13;
int ledState = LOW;
long previousMillis = 0;
long interval = 2000;

void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(115200);
}

void loop()
{
  int val = analogRead(ANALOG_IN);
  writeOscilloscope(val);
  unsigned long currentMillis = millis();
  if(currentMillis - previousMillis > interval) {
    previousMillis = currentMillis;
    if (ledState == LOW)
      ledState = HIGH;
    else
      ledState = LOW;
    digitalWrite(ledPin, ledState);
  }
}
```

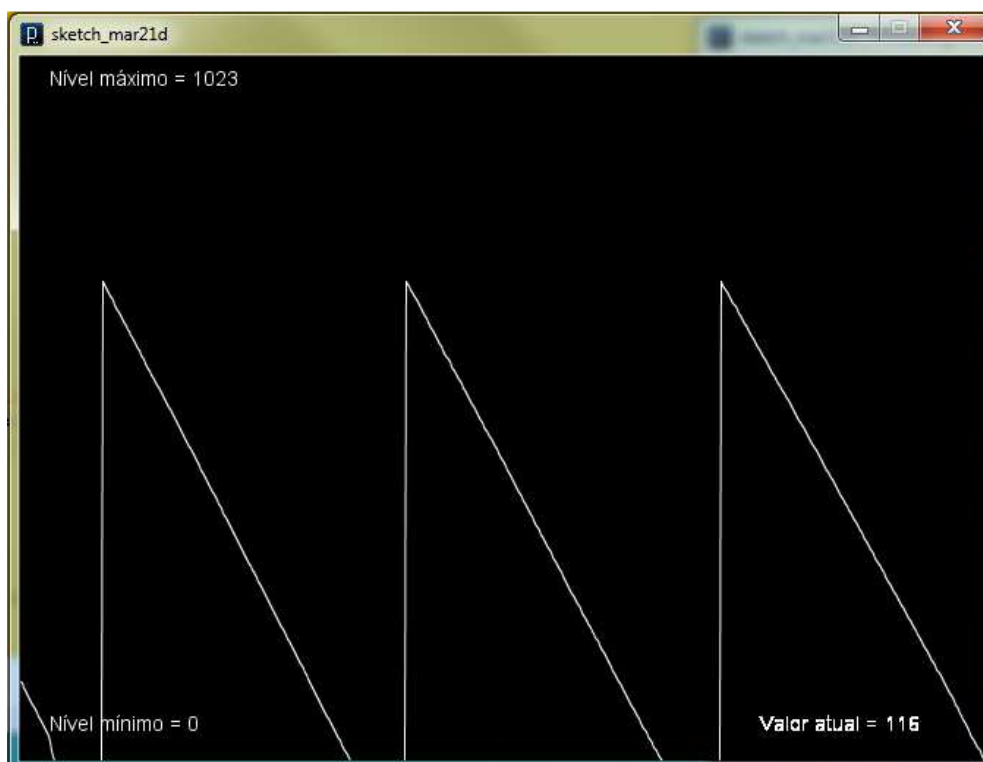
Este código é muito semelhante ao anterior que gravamos no Arduino. A grande diferença entre este e aquele último é que, quando usamos o 555 no modo de Rampa Linear, precisamos dar um pulso em uma entrada do 555 para que a rampa comece a ser desenhada na saída do CI.

Veja que agora a saída do CI não é no pino 3, e sim nos pinos 6 ou 7 (veja por aí a versatilidade deste chip). Deste modo, temos que de algum modo dar este pulso usando o Arduino, e é por isso que ligamos o pino 2 do 555 no pino 13 do Arduino. O pino 2 do 555 é conhecido como **trigger**, ou seja, gatilho. Nesta configuração, quando houver um pulso em nível lógico baixo, a rampa começará a nascer. Desta forma, via código no Arduino deixamos o pino 13 em baixa durante 2 segundos e depois deixamos ele em alta por 2 segundos. Durante estes 2 segundos que o deixamos em alta, podemos ver a rampa crescendo no [Processing](#).

Toda vez que houver nível lógico baixo (ou seja, LED desligado na placa pois estamos usando o pino 13), começará uma nova rampa. Desta forma, poderíamos ter usado um código como o blink, que liga o pino, deixa um tempo ligado, depois desliga, deixa um tempo desligado e assim por diante. Porém, quando usamos o código blink, usamos os **delays**, comandos estes que deixam o código "travado". No código acima, para não usar **delays**, usamos a rotina de **millis()**. Esta rotina nos permite executar funções de tempos em tempos sem travar o programa. Ele fica rodando tudo o que tem na rotina **loop()** e quando o intervalo é de 2 segundos (por causa da variável **interval**) ele executa o que existe na rotina condicional, ou seja, liga e desliga o pino 13.

Pode parecer um pouco complicado, porém se você quiser estudar isto mais a fundo (o que é uma enorme utilidade quando se faz códigos), veja o exemplo que existe no programa Arduino, indo em **File > Examples > 02.Digital > BlinkWithoutDelay**.

Se tudo estiver certo com nosso circuito, e após o código do Arduino ser gravado na placa, quando rodarmos nosso código no [Processing](#), devemos ver o seguinte:



Novamente, a forma de onda esperada foi obtida: uma Rampa Linear!

O software **Processing** é extremamente poderoso e o que fizemos aqui é uma parcela muito pequena do que ele é capaz. Com o Processing ainda seria possível fazer sistemas supervisórios para controle de processos, painéis para leitura de sensores e acionamento de periféricos com o Arduino e por aí vai. Iremos usar mais o **Processing** nesta apostila, porém se você se interessa pela parte gráfica acesse <http://www.Processing.org> e veja tutoriais, documentos para referência de código, exemplos, etc.

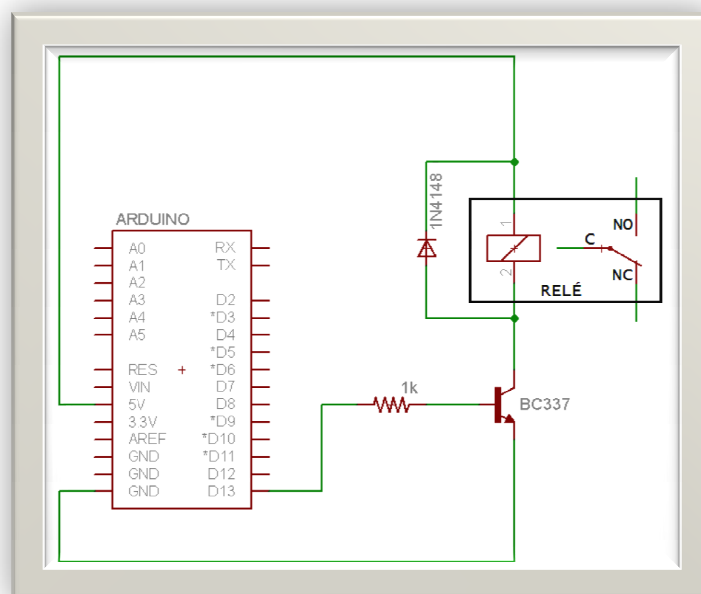
- **Acionamento de Cargas com Módulo Relé**

Componentes: 01xMódulo Relé

Descrição: Aprenda como acionar cargas de maior porte. Com este experimento será possível acionar e desacionar qualquer carga cuja especificação esteja dentro das especificações do relé presente no módulo.

Este provavelmente é um dos projetos mais executados por estudantes e hobbistas, que admiram a arte da eletrônica. O simples uso de um pequeno sinal de corrente contínua de 5V, para acionar uma lâmpada de 110V, por exemplo. Pense nas possibilidades. Você pode construir seu próprio projeto de domótica, ou seja, automatizar toda uma residência com uma simples placa Arduino. Como? Que tal colocar o controle de sua casa inteira na internet, e acessá-lo por um site cuja senha e login só você tem? Isto lhe permitiria verificar se deixou alguma luz acesa em casa quando saiu para passear, ou acender uma luz a noite quando a casa estiver sozinha para ter mais segurança. Pense nas possibilidades de poder ativar ou desativar qualquer aparelho, estando a quilômetros de distância de sua casa. Você pode fazer isso de diversos modos, seja com um Shield Ethernet e um cabo ethernet com conexão à internet, ou um Shield WiFi (que se conecta a um roteador wireless com internet), ou mesmo utilizando um Shield Celular (ou GSM), e fazer todo o controle de ambientes via mensagens SMS. Ok, se domótica não for bem seu interesse, você pode acionar relés afim de controlar a movimentação de um robô, com as velhas e boas pontes H feitas de relé.

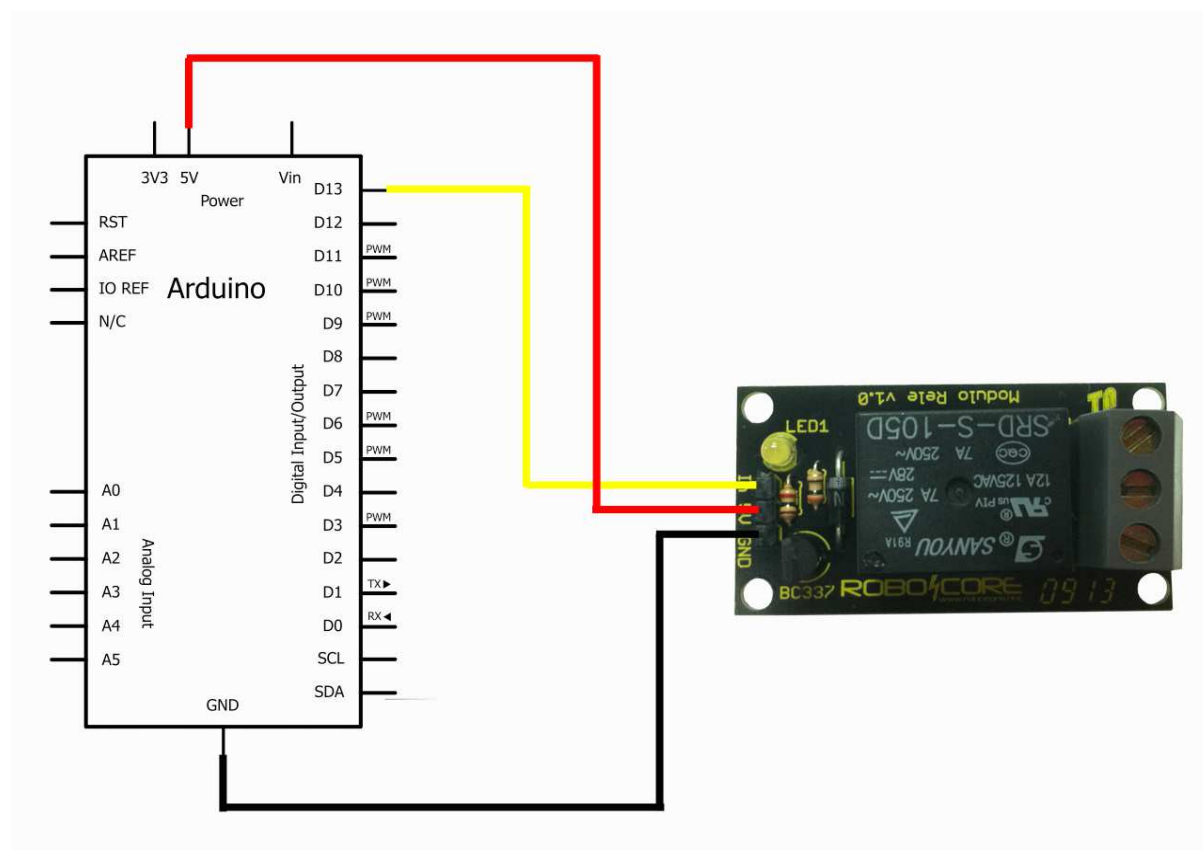
Chega de falar sobre o que pode ser feito, e vamos fazer. Acionar um relé com Arduino não é uma tarefa difícil. Um relé nada mais é que um componente que, através de condução eletromagnética faz uma chave se movimentar. Portanto, um relé é simplesmente uma chave liga e desliga! Normalmente um relé possui 5 terminais. Dois deles são os terminais de uma bobina de acionamento. Esta bobina, no relé que está na plaquinha que acompanha este kit, é de 5V. Ou seja, você precisa de uma tensão DC de 5V para fazer a bobina conduzir e fazer o “relé bater”, ou acionar a chave. Os outros três terminais são como de uma simples chave, um é o terminal comum, ou seja, ele é comum ao circuito. Nele sempre passará a corrente. Os outros dois terminais são os contatos, um é normalmente aberto (NA) e outro é normalmente fechado (NF). Eles são contatos com funções cujo próprio nome está dizendo. O normalmente fechado está em contato com o pino COMUM a todo momento **até que** a bobina 5V conduza corrente. Quando a bobina conduz corrente, o contato do relé se fecha, e o pino COMUM se liga ao pino NORMALMENTE ABERTO, fazendo-o ficar fechado e conduzir a corrente. Apenas para ilustrar, basicamente para acionar um relé com Arduino precisamos do seguinte circuito:



No módulo relé que acompanha o kit ainda existe um circuito a mais que faz um LED acender quando o relé está conduzindo.

Mas porque não podemos simplesmente ligar a bobina do relé em uma saída digital do Arduino? É necessário utilizar um transistor para acionar a bobina do relé, pois a corrente que sai das portas digitais da placa não é suficiente para excitá-la. Ligando diretamente você corre o risco de queimar a porta da placa.

Utilizando três cabos Macho/Fêmea que acompanha o kit, faça a seguinte ligação:



Um código simples para ver se o circuito está funcionando, é o exemplo mais básico do Arduino, o *blink*:

```
/*
 * Blink
 * Turns on an LED on for one second, then off for one second, repeatedly.
 *
 * This example code is in the public domain.
 */

void setup() {
  // initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}

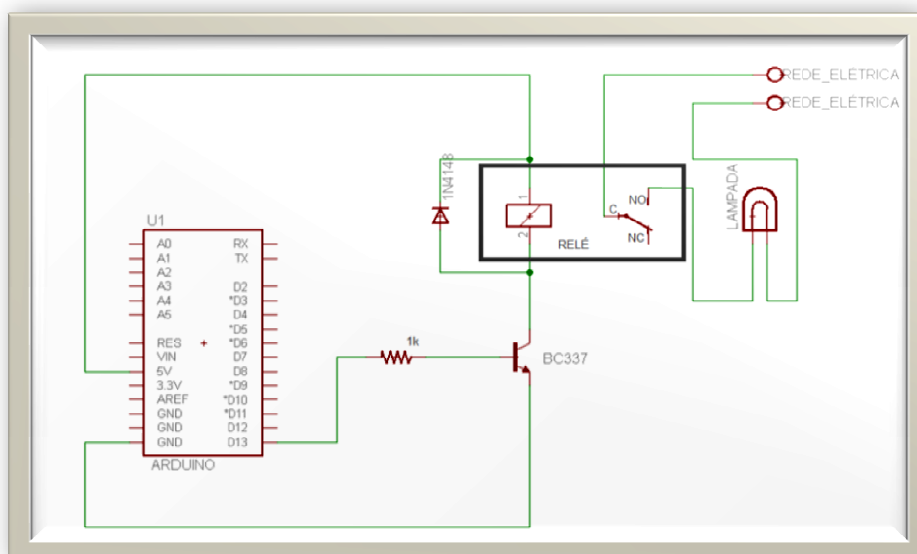
void loop() {
  digitalWrite(13, HIGH); // set the LED on
  delay(1000);            // wait for a second
  digitalWrite(13, LOW);  // set the LED off
  delay(1000);            // wait for a second
}
```

Caso você ainda não tenha ciência disto, o código *blink* pode ser encontrado em

FILE > EXAMPLES > 1. BASICS > BLINK

Se tudo der certo, você deverá ouvir o ruído de uma chave abrindo e fechando, no caso, o contato do relé. Caso não esteja ouvindo este ruído, revise o circuito e verifique se o código foi gravado corretamente.

Pronto! O relé já está funcionando e agora sua imaginação é o limite. Veja abaixo um exemplo de circuitos utilizando relé para acionar cargas:



NOTA:

A RoboCore não se responsabiliza por danos à rede elétrica e/ou à integridade física do praticante do experimento, bem como quaisquer pessoas que possam vir a ser lesadas por estes experimentos.

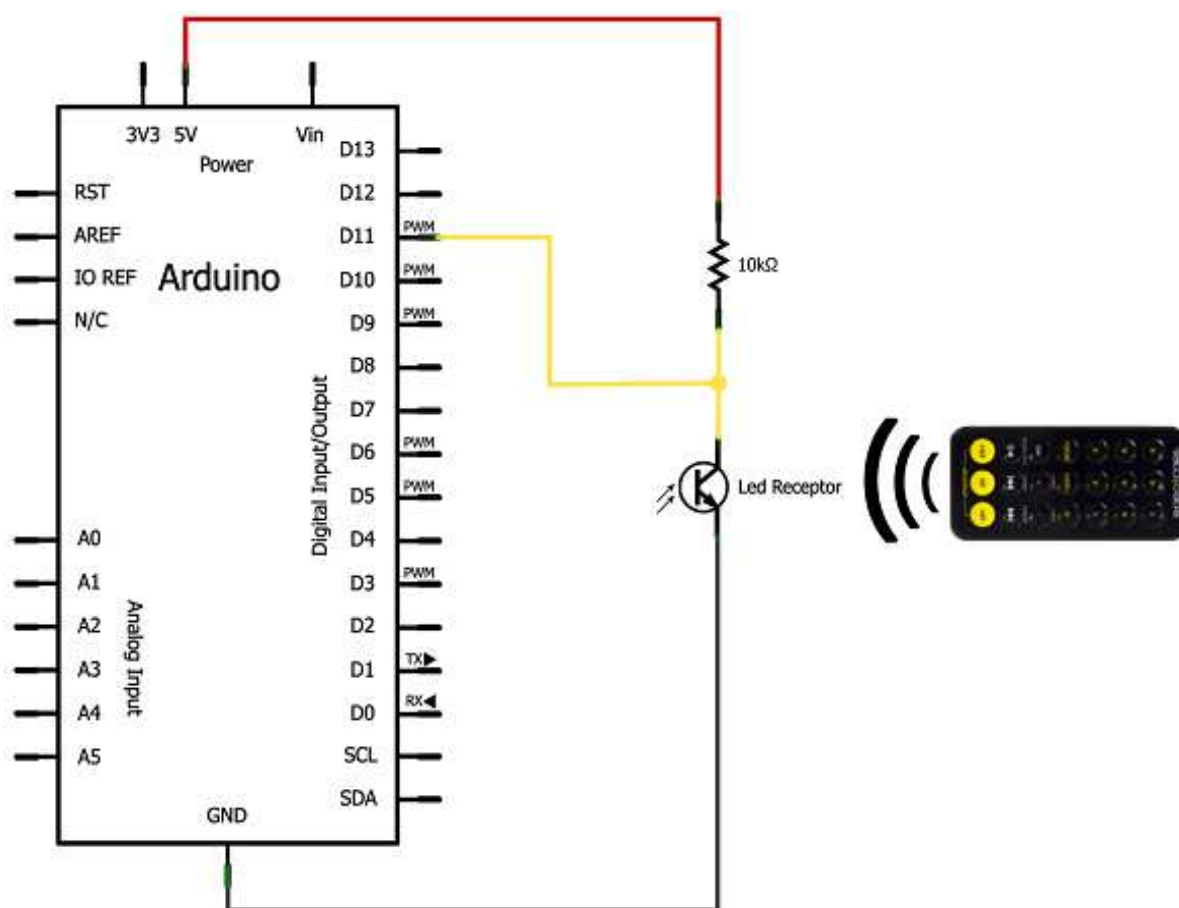
Se você não sabe ou não tem certeza do que está fazendo, procure por maiores informações e, principalmente, não faça nada que você não tenha certeza!

- **Controle Remoto**

Componentes: 01xControle Remoto + 01x LED Receptor Infravermelho + 01xMódulo Relé
Descrição: Aprenda como utilizar a maioria dos controles remotos com Arduino, e no final aprenda a ligar e desligar uma carga usando o controle.

Após a sua invenção, o controle remoto passou a ser um item encontrado em praticamente qualquer domicílio. É realmente difícil encontrar um lugar em que as pessoas ainda se levantam de seus sofás para mudar o canal, aumentar o volume, trocar a música do aparelho de CD, dar o play no Blu-Ray ou DVD, e por aí vai. Controles remotos estão presentes em todos os lugares e todos tem o mesmo princípio de funcionamento: o envio de informações criptografadas via raios infravermelhos. Isto parece algo extremamente complicado, porém com Arduino isto será desmistificado. No início deste experimento, iremos usar o controle que vai junto no kit, porém sintase a vontade em usar algum que você tiver em sua casa.

O primeiro passo deste experimento é colocar a bateria CR2025 no controle remoto. Então, ligue o receptor de raios infravermelhos, ou seja, o LED receptor IR (fototransistor) no Arduino. O LED que iremos utilizar neste experimento é um dos LEDs que usamos no experimento sobre detecção de obstáculos através de LEDs detectores e receptores infravermelho. Neste experimento, iremos utilizar o LED que possui um pequeno ponto na cor **vermelha**. Para começar, monte o seguinte circuito utilizando o Arduino e o LED:



Feito a montagem, iremos colocar o código no Arduino. Para este experimento, iremos utilizar uma biblioteca chamada *"IRremote.h"*. Esta biblioteca encontra-se na pasta **Bibliotecas** no CD que acompanha este kit. Neste manual você já aprendeu como instalar corretamente bibliotecas na IDE do Arduino, portanto faça aquele mesmo procedimento para esta biblioteca.

Após colocar a pasta da biblioteca no local apropriado, coloque o seguinte código na IDE:

```
#include <IRremote.h>

int RECV_PIN = 11;
IRrecv irrecv(RECV_PIN);
decode_results results;

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

void loop() {
  if (irrecv.decode(&results)) {
    Serial.println(results.value, HEX);
    irrecv.resume();
  }
}
```

O código acima é uma adaptação do exemplo **IRrecvDemo** presente na própria biblioteca. O código simplesmente recebe informações do LED receptor (fototransistor), que está ligado ao pino 11, e mostra no monitor serial seu código em formato HEXADECIMAL. Após gravar o código acima no Arduino, abra o Monitor Serial e aperte as teclas do controle remoto que acompanha o kit. Pressionando o botão **CH-** você deve receber o valor **FFA25D**. Cada botão do controle remoto possui um código hexadecimal diferente. Pressione todos os botões para ver cada um dos códigos. Caso tenha curiosidade, pegue outros controles remotos e faça o mesmo teste. Caso o controle seja compatível com esta biblioteca, será mostrado um código em hexadecimal. Caso contrário, irá aparecer um número zero (0) cada vez que você pressionar um botão. As vezes, ao apertar um botão do controle que acompanha este kit, você pode receber o número zero, porém basta pressionar novamente que irá aparecer o código correto.

Atenção: caso você esteja recebendo o seguinte erro ao tentar gravar o programa na placa:

C:\Program Files (x86)\ArduinolibrariesRobotIRremotesrc\IRremoteTools.cpp:5:16: error: 'TKD2' was not declared in this scope

```
int RECV_PIN = TKD2; // the pin the IR receiver is connected to
```

Resolva este erro **removendo** da pasta libraries, a pasta **RobotIRremote**.

Conhecer o código de cada tecla é muito útil. Desta forma, você pode fazer uma programação condicional em seu Arduino que faça com que, para cada botão pressionado algo diferente aconteça. Em nosso exemplo, iremos utilizar o botão com o número 1 do controle para acionar um relé, e o botão de número 2 para desligar este relé. Para isto, primeiramente devemos fazer o teste com o Arduino, sem ligar a parte do relé. Abra o monitor serial e pressione o número 1 do controle remoto. Você deverá receber o seguinte código em hexadecimal: **FF30CF**. Pressione agora o número 2 do controle. Você deverá receber o seguinte código: **FF18E7**.

Dica: Para que você tenha um resultado mais satisfatório neste experimento, mire a frente do controle remoto diretamente para a parte abaulada do LED receptor infravermelho. Uma boa distância entre controle e receptor é de 5cm. O receptor não conseguirá receber os dados enviados pelo controle por distâncias maiores por causa do LED receptor. Em aparelhos domésticos é utilizado um outro tipo de LED receptor, o qual consegue fazer leituras a alguns metros. De toda forma, para testar o conceito você pode utilizar este LED. Caso queira algo mais distante você pode adquirir um LED especial para isto por um valor baixo.

Agora que sabemos os códigos que os números 1 e 2 enviam quando apertamos os botões no controle, precisamos apenas pensar em um código para o Arduino que quando receba aqueles valores em hexadecimal faça o que queremos.

Grave o seguinte código no Arduino:

```
#include <IRremote.h>

const int RECV_PIN = 11;
IRrecv irrecv(RECV_PIN);
decode_results results;

void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn(); //começa a receber
}

void loop() {
  if (irrecv.decode(&results)) {
    Serial.println(results.value, HEX);

    if (results.value == 0xFF30CF){
      Serial.println("Botao 1 Pressionado");
    }

    if (results.value == 0xFF18E7){
      Serial.println("Botao 2 Pressionado");
    }

    irrecv.resume(); // Recebe o próximo valor
  }
}
```

Apenas ao observar o código, você entende o que deve acontecer?

Quando pressionarmos o botão 1, o código recebido será o que já vimos anteriormente. Devemos então ver no monitor serial os dizeres *"Botao 1 Pressionado"*. Veja que na condicional o valor em hexadecimal deve ser colocado com um **"0x"** antes. Isto serve para a formatação em hexadecimal do valor recebido. Ao pressionar o número 2, devemos ver no monitor serial os dizeres *"Botao 2 Pressionado"*. A ideia de fazermos um teste sem ligar diretamente o relé serve para prevenirmos erros. Sempre que for utilizar cargas, teste primeiro sem a carga, apenas pelo monitor serial. Quando você tiver certeza que tudo está funcionando, adicione o hardware da carga.

Se tudo estiver funcionando, ligue o **IN** do módulo Relé no pino **12** do Arduino, o GND do módulo a um GND do Arduino e o 5V do módulo ao 5V do Arduino. Feita a ligação, grave o seguinte código no Arduino:

```
#include <IRremote.h>

const int RECV_PIN = 11;
IRrecv irrecv(RECV_PIN);
decode_results results;
const int rele = 12;

void setup()
{
  pinMode(rele, OUTPUT);
  digitalWrite(rele, LOW);
  Serial.begin(9600);
  irrecv.enableIRIn(); //começa a receber
}

void loop() {
  if (irrecv.decode(&results)) {
    Serial.println(results.value, HEX);
  }

  //continua na proxima pagina
}
```

```
//continacao:

if (results.value == 0xFF30CF){
    Serial.println("liga");
    digitalWrite(rele, HIGH);
}

if (results.value == 0xFF18E7){
    Serial.println("desliga");
    digitalWrite(rele, LOW);
}

    irrecv.resume(); // Recebe o próximo valor
}
```

Este é apenas um dos exemplo, pense nas possibilidades que você pode fazer com este controle.

Agora vamos mergulhar na leitura de dois sensores realmente muito interessantes, são eles o **Acelerômetro** e o **Sensor Ultrassônico**!

- **Acelerômetro**

Componentes: 01x Acelerômetro + 02x Resistores de 300Ω

Descrição: Aprenda como funciona este sensor, que está presente na maioria dos dispositivos móveis da atualidade, como smartphones, tablets e até notebooks.

Acelerômetros hoje estão presentes na maioria dos dispositivos móveis. Eles servem, de uma forma bem ampla, para verificar a posição que o dispositivo está e também para verificar se está em queda livre. Sabendo a posição que o dispositivo está é possível rotacionar as informações na tela para serem melhor visualizadas pelo usuário do smartphone ou tablet. Uma função que torna o uso deste tipo de sensor muito importante é a verificação de queda livre. Hoje, em laptops mais avançados, existem acelerômetros que conseguem verificar se o computador está caindo e com isso conseguem desligar o disco de dados e, em um eventual choque com o chão, o disco pode ser recuperado já que não foi desligado abruptamente durante o choque. Além de aplicações como estas, acelerômetros também estão presentes em robôs do tipo *segway*, *quadrópteros*, etc. Basicamente servem para encontrar a posição dos mesmos através da aceleração da gravidade (por isso chama-se acelerômetro). Atualmente é muito comum encontrar acelerômetros que medem os 3 eixos em relação ao solo, porém antigamente era comum contrair com 2 eixos. O acelerômetro presente neste kit é o MMA8452Q. Ele possui 3 eixos para leitura e é digital, ou seja, para usá-lo usaremos pinos digitais da placa Arduino.

O chip MMA8452Q irá se comunicar com o Arduino de forma digital. As placas Arduino possuem algumas formas de comunicação digital, como SPI e I²C. Neste caso, usaremos a comunicação I²C. Este tipo de comunicação foi desenvolvido pela Philips e é muito interessante pois permite utilizar diversos periféricos ou sensores com apenas dois pinos da placa. Os pinos usados são o SDA e o SCL. Apenas para se ter uma ideia de como funciona isso de se ter mais de um periférico ligado ao mesmo pino no Arduino, para se comunicar com cada um deles o Arduino sabe o endereço hexadecimal dos mesmos, e antes de requerir dados dos sensores, antes ele envia pela linha de comunicação o endereço de quem ele quer "conversar". Como cada dispositivo tem seu endereço, apenas aquele com o endereço solicitado irá "responder" à solicitação. Para fins de conhecimento vale dizer que SDA vem de *Serial Data* e SCL vem de *Serial Clock* e já que falamos de endereçamento, vale dizer que o endereço do acelerômetro que iremos usar é **0x1C** (porém, caso seja necessário usar dois chips idênticos na mesma placa, existe a opção de alterar o endereço fazendo uma alteração na parte inferior da placa, no caso uma mudança no hardware irá alterar algo no software). Não se preocupe com este endereço, iremos utilizar uma biblioteca feita pela **SparkFun**, mesma empresa que fabricou a plaquinha que o chip do acelerômetro está soldada, e o endereço estará dentro da biblioteca e não precisaremos fazer alterações.

Para começar este experimento, então, instale a biblioteca chamada SFE_MMA8452Q que está presente na pasta **Bibliotecas** do CD em seu programa Arduino, assim como fizemos para as bibliotecas *oscilloscope* e *IRremote*. Feita a instalação, você deverá ver o nome da biblioteca se acessar o caminho:

Sketch > Import Library... > SFE_MMA8452Q

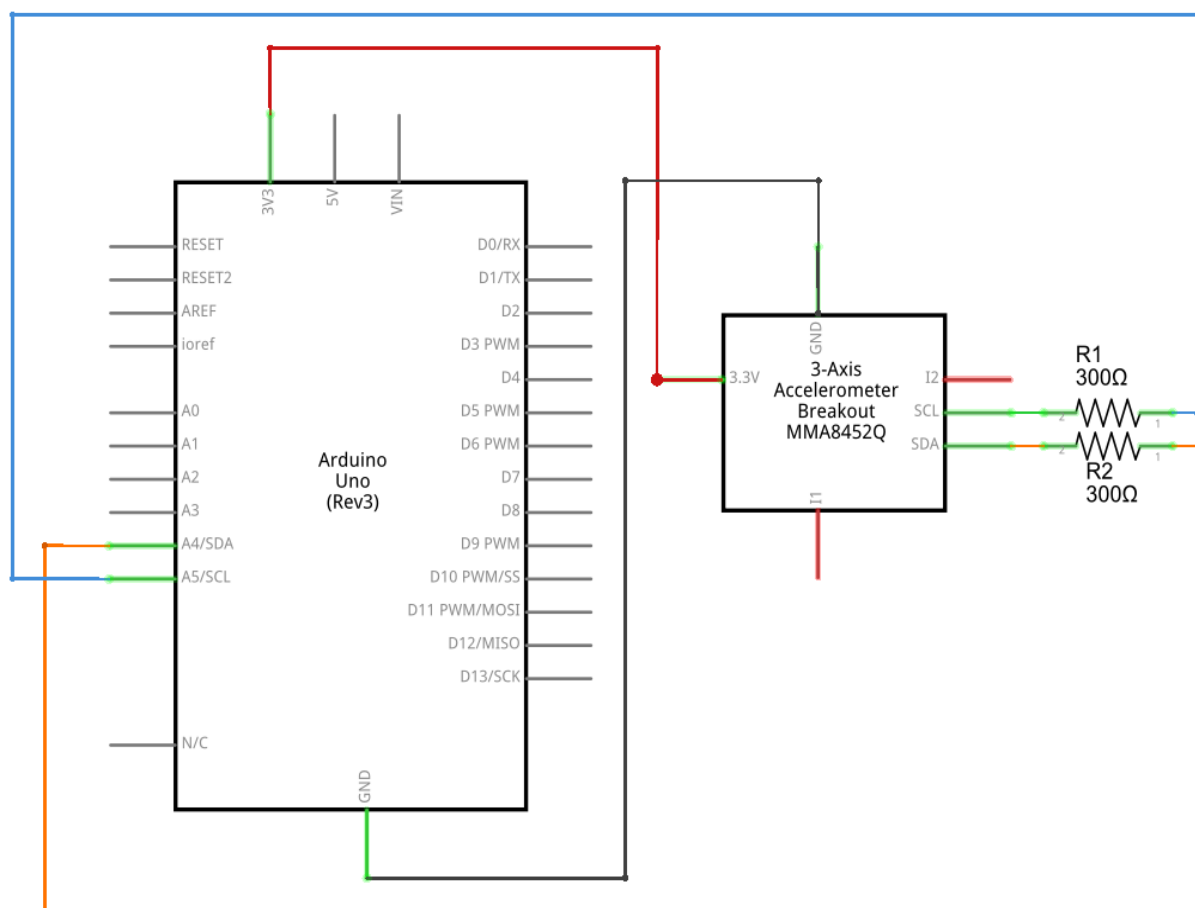
Novamente, não é necessário clicar nela. Apenas de verificar que ela está lá já é o suficiente.

O próximo passo será montar o circuito. Diferente de todos os módulos que usamos neste manual e também diferente de todos que usamos no Kit Iniciante para Arduino, este sensor deve ser alimentado com **3,3V**. Tome **muito** cuidado com isso, pois se ligar em 5V pode danificar o componente permanentemente e a garantia não cobre este tipo de problema. Com muita atenção e cuidado, verifique na placa quem são os pinos e faça a ligação de 4 pinos conforme o esquema a seguir. Veja que você deverá ligar o pino de 3,3V do sensor no pino de 3,3V da placa Arduino (está próximo ao pino 5V, o que exige muito mais atenção na hora de ligar). Ligue o GND do sensor no GND da placa. Então ligue um resistor de 300Ω no pino SCL e um outro resistor de 300Ω no SDA do sensor. Estes resistores servem para baixar a tensão de 5V da placa para 3,3V para o sensor. Fazemos isso principalmente para não injetar uma tensão maior que 3,3V no sensor. A ligação do **outro lado** de cada resistor na placa depende de qual placa Arduino você está usando. Na figura

abaixo está sendo mostrado como ligar no Arduino UNO, porém os pinos mudam para o Arduino Leonardo e também para o Arduino Mega 2560 R3. Veja abaixo onde ligar em cada uma das placas:

TIPO DE PLACA	Pino SDA	Pino SCL
Arduino UNO R3 & BlackBoard	Analogico 4	Analogico 5
Arduino Leonardo	Digital 2	Digital 3
Arduino Mega 2560 R3	Digital 20	Digital 21

Porém, existe outra forma de usar os dois pinos da comunicação I²C que é comum para todas as placas mais recentes (placas após a revisão 3, ou R3). No barramento onde existem os pinos digitais de 8 a 13, logo depois existem os pinos **GND** e **AREF**. Logo depois do pino AREF existem dois pinos. Na ordem, eles são o SDA e depois o SCL. Faça a ligação onde você achar melhor. Outra dica é colocar o sensor na protoboard e a placa Arduino sobre a mesma protoboard, presa na mesa com um elástico. Isso fará com que mexer no conjunto para mudar a posição do sensor fique bem mais prático. O esquema do circuito é:



Novamente, isso não é uma competição então não tenha pressa para montar o circuito. Faça as ligações com bastante atenção para não ter perigo nem de queimar o acelerômetro nem de ligar algo errado na parte de comunicação.

Feita a ligação, grave o seguinte código no Arduino:

```
#include <Wire.h>
#include <SFE_MMA8452Q.h>

MMA8452Q accel;

void setup(){
  Serial.begin(9600);
  Serial.println("MMA8452Q Test Code!");
  accel.init();
}

void loop(){
  if (accel.available()) {
    accel.read();
    printCalculatedAccels();
    printOrientation();
    Serial.println();
  }
}

void printAccels(){
  Serial.print(accel.x, 3);
  Serial.print("\t");
  Serial.print(accel.y, 3);
  Serial.print("\t");
  Serial.print(accel.z, 3);
  Serial.print("\t");
}

void printCalculatedAccels(){
  Serial.print(accel.cx, 3);
  Serial.print("\t");
  Serial.print(accel.cy, 3);
  Serial.print("\t");
  Serial.print(accel.cz, 3);
  Serial.print("\t");
}

void printOrientation(){
  byte pl = accel.readPL();
  switch (pl)
  {
    case PORTRAIT_U:
      Serial.print("Portrait Up");
      break;
    case PORTRAIT_D:
      Serial.print("Portrait Down");
      break;
    case LANDSCAPE_R:
      Serial.print("Landscape Right");
      break;
    case LANDSCAPE_L:
      Serial.print("Landscape Left");
      break;
    case LOCKOUT:
      Serial.print("Flat");
      break;
  }
}
```

O código acima nada mais é que o código de exemplo da biblioteca da **SparkFun** para o acelerômetro que estamos usando, porém sem os comentários. Você pode encontrar o código completo, com comentários explicando o mesmo bem como com as demais funções que o código pode ter clicando em **File > Examples > SFE_MMA8452Q > MMA8452Q_Basic** (inclusive, se você entende inglês, seria bastante interessante dar uma olhada nos comentários do código original pois trata-se de um ótimo trabalho para usar este acelerômetro).

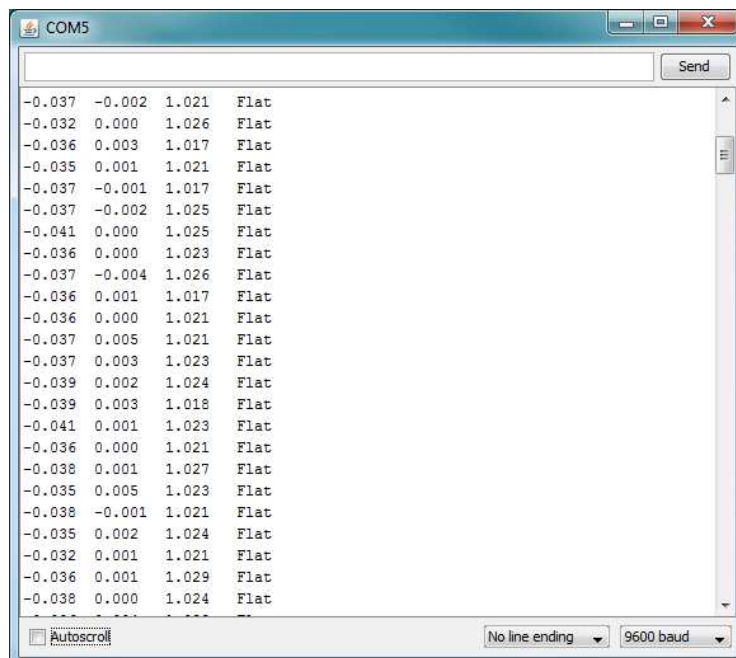
Antes de mais nada, vamos tentar entender o que o código faz. Nas duas primeiras linhas invocamos as bibliotecas **Wire** e **SFE_MMA8452Q**. Esta última biblioteca já sabemos que serve para fazer as leituras do acelerômetro, porém é a primeira vez que vemos a biblioteca **Wire**. Esta biblioteca é usada pelo Arduino para fazer comunicação utilizando I²C. Ela facilita o uso dos pinos SDA e SCL junto aos periféricos. No restante do código aparentemente não são usadas funções referentes a esta biblioteca, porém dentro da biblioteca **SFE_MMA8452Q** são usadas as funções necessárias para a comunicação I²C.

Depois de "chamar" as bibliotecas que iremos usar, criamos um objeto chamado *accel* que é da classe **MMA8452Q** (caso você tenha curiosidade em entender mais sobre classes existem ótimos materiais disponíveis sobre a linguagem C++, dominando esta linguagem é possível fazer muito mais que códigos para Arduino, mas também bibliotecas, além de muitas outras coisas).

No **setup** do programa, iniciamos a comunicação serial com taxa de 9600bps, isto servirá para olharmos o que o sensor esta lendo no monitor serial, bem por isso na próxima linha de código é escrito no monitor serial a frase **MMA8452Q Test Code!** Então o objeto *accel* é iniciado.

No **loop** do programa fazer as leituras propriamente ditas se houverem dados disponíveis para leitura. Se houverem tais dados, entramos em duas funções. Uma delas é chamada **printCalculatedAccels()** e a outra é a **printOrientation()**. Então no final do **loop** simplesmente escrevemos um caracter vazio, porém com o **println** para irmos para a próxima linha. Nestas duas funções são imprimidas no monitor serial o valor do acelerômetro nas três componentes, ou seja, em relação a **x**, **y** e **z** e também qual é a posição atual do acelerômetro. Toda a lógica por trás disso está dentro da biblioteca. A partir dos valores das componentes **x**, **y** e **z** é possível saber se o sensor está na posição normal, se está virado para cima, para baixo ou para um dos lados.

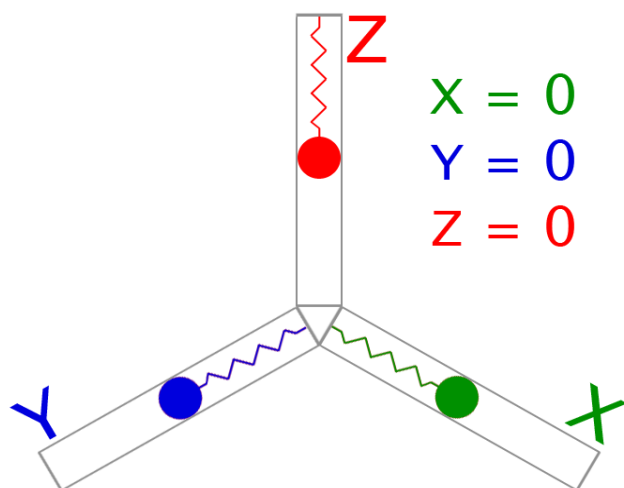
Vamos agora abrir o monitor serial para verificar o que temos de valores. Vemos o seguinte com a protoboard parada sobre uma superfície:



A última informação da linha é fácil de entender, ou seja, recebemos um *Flat* se a protoboard com o sensor está deitada na superfície. Agora experimente virar a protoboard para a direita, depois para esquerda, para cima e para baixo, ou seja, vire a protoboard para que uma das superfícies mais finas fique em contato com a protoboard. Você deverá ver os valores numéricos alterarem, e a palavra final mudará para uma das seguintes: *Portrait Up*, *Portrait Down*, *Landscape Right* ou

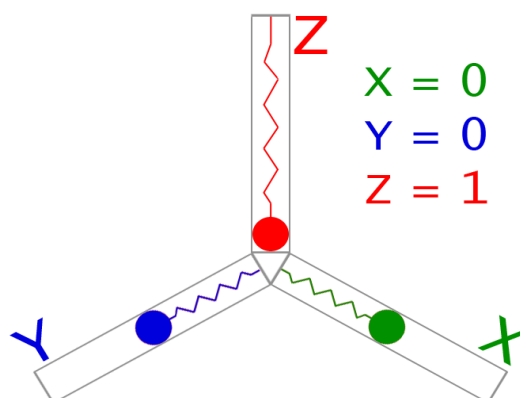
Landscape Left. Todas essas informações o programa só sabe devido a mudança nos números das componentes **x**, **y** e **z**. Mas então, o que está acontecendo com as três componentes?

Para entender o que está acontecendo, e o que são estes números vamos dar uma olhada na seguinte imagem:



Podemos imaginar um acelerômetro na prática como sendo três massas presas por "molas" (sem resistência) em três pontos fixos. Cada um dos eixos está perpendicular em relação aos outros. As três massas excursionam livremente de cima para baixo, porém elas não vão para os lados (imagine como se estivessem dentro de um tubo e o único movimento que conseguem fazer é este, de cima para baixo e vice-versa). Quando a massa estiver no **centro** do tubo, a leitura será **zero**, pois a "mola" não está nem esticada para baixo, nem comprimida para cima. Quando a massa estiver "puxada" para baixo, já que a gravidade "puxa" a massa para baixo, o valor vai até 1. E quando viramos o sistema de ponta cabeça, a gravidade "puxa" a massa para o lado que está fixada, então o valor vai até -1.

Agora, dê uma olhada na imagem ao lado. Trata-se da situação atual de nosso sensor, ou seja, na posição que verificamos no monitor serial a palavra *FLAT*. Neste caso, com o sensor repousando sobre a superfície, apenas a massa do eixo **Z** está deslocada para baixo, já que a gravidade está "puxando" a massa para baixo. No caso, temos o valor de **1**, conforme podemos também ver no monitor serial. De forma análoga, podemos pensar que, se virarmos a protoboard que o sensor está de ponta cabeça, a massa ficará encostada na superfície onde está presa, logo o valor desta componente será **-1**.



Agora, experimente virar a protoboard onde está o sensor, de modo que o lado onde existe a barra de pinos soldada fique para baixo. Se olharmos a própria plaquinha podemos ver que existem as letras **X**, **Y** e **Z**. Se pararmos e analisarmos a situação, podemos presumir que a componente em **Y** e **Z** agora não terão nenhuma participação, já que apenas a massa da componente **X** está sendo "puxada" para baixo. Dessa forma, podemos observar no monitor serial a seguinte informação:

```
1.008 0.014 -0.012 Landscape Right
1.009 0.013 -0.007 Landscape Right
1.006 0.013 -0.014 Landscape Right
1.008 0.015 -0.005 Landscape Right
1.011 0.013 -0.007 Landscape Right
1.012 0.014 -0.006 Landscape Right
1.006 0.012 -0.009 Landscape Right
```

Ou seja, temos a componente **X** valendo 1 (ou algo próximo disso) e as outras componentes valendo 0 (ou algo próximo disso). Com as informações das componentes, o programa consegue identificar a posição que o sensor está virado, se é na posição normal, para cima, para baixo, direita ou esquerda. Apenas um detalhe nessa explicação, este 1 ou -1 que estamos recebendo refere-se a quantidade de **g** que temos naquele momento, ou seja, como a força gravitacional está agindo sobre

o sensor. Se você tiver curiosidade, tente mover rapidamente, de forma brusca, a protoboard com o sensor para um dos lados e ao mesmo tempo desmarque a opção *Autoscroll* da tela do Monitor Serial. Você verá que durante a movimentação o número pode ser maior que 1 (ou -1). Como o sensor está configurado para leituras de $\pm 2g$, você consegue inclusive verificar quão rápido foi a mudança de direção. Este sensor pode fazer leituras de $\pm 2g$, 4g ou 8g, dependendo da precisão que for necessária. Agora, volte na primeira figura da página anterior, onde exemplificamos como poderíamos entender o acelerômetro. Naquele caso, as três massas estão no meio dos "tubos". Você consegue imaginar em qual situação teríamos as três massas no meio, recebendo os três valores das três componentes iguais a **zero**? Se você está pensando no momento de queda livre, você acertou. Quando o dispositivo está em queda livre, ou seja, o corpo não está sobre uma superfície, não existe uma força contrária à força do dispositivo, como a força Normal (para saber mais, pesquise sobre a terceira Lei de Newton, sobre ação e reação). Então, as massas ficam livres dentro do corpo e ficam na posição onde o valor de cada componente dá **0** (só que não vamos querer provar este conceito derrubando nosso sensor no chão, certo?)

Agora, apenas para fixar bem o conceito, iremos visualizar graficamente no computador, com a ajuda do software **Processing**, para qual lado o sensor está virado, conforme as posições que vimos no monitor serial. Para isso, grave o seguinte código no Arduino:

```
#include <Wire.h>
#include <SFE_MMA8452Q.h>

MMA8452Q accel;

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

void loop()
{
  if (accel.available()){
    accel.read();
    printOrientation();
    delay(250);
  }
}

void printOrientation()
{
  byte pl = accel.readPL();
  switch (pl)
  {
    case PORTRAIT_U:
      Serial.print(0);
      break;
    case PORTRAIT_D:
      Serial.print(1);
      break;
    case LANDSCAPE_R:
      Serial.print(2);
      break;
    case LANDSCAPE_L:
      Serial.print(3);
      break;
    case LOCKOUT:
      Serial.print(4);
      break;
  }
}
```

O código acima é uma adaptação do código anterior. Veja que agora não estamos mais interessados em saber o valor de cada componente, e sim em saber em qual posição o sensor está.

Para cada posição a placa Arduino irá enviar via cabo USB para o computador um número. Caso o sensor esteja na posição *Portrait Up*, a placa enviará um número 0 para o computador. Se estiver na posição *Landscape Left*, enviará um número 3 e assim por diante. Com o código gravado no Arduino, coloque o seguinte código no **Processing** e clique no botão **RUN**.

```
import processing.serial.*;
Serial port;
int val;
char valor;

void setup() {
  println(Serial.list());
  port = new Serial(this, Serial.list()[0], 9600);
  size(600, 600);
  rectMode(CORNERS);
}

void draw() {
  if (port.available() > 0) {
    val = port.read(); //recebo a variavel em decimal que representa um char conforme ASCII
    valor = char(val); //transformo em char
  }
  switch(valor){
    case '0':
      up(); //seta para cima
      break;

    case '1':
      down(); //seta para baixo
      break;

    case '2':
      right(); //seta para direita
      break;

    case '3':
      left(); //seta para esquerda
      break;

    case '4':
      flat(); //posicao normal
      break;
  }
}

void flat() {
  clear();
  ellipse(300, 300, 400, 400);
}

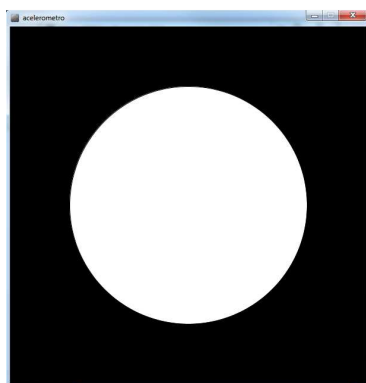
void up() {
  clear();
  rect(200, 600, 400, 300);
  triangle(300, 0, 0, 300, 600, 300);
}

void down() {
  clear();
  rect(200, 0, 400, 300);
  triangle(0, 300, 600, 300, 300, 600);
}

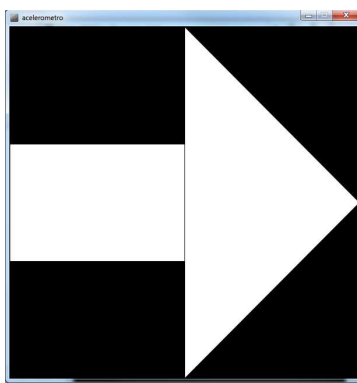
void right() {
  clear();
  rect(0, 200, 300, 400);
  triangle(300, 0, 600, 300, 300, 600);
}

void left() {
  clear();
  rect(300, 200, 600, 400);
  triangle(300, 0, 300, 600, 0, 300);
}
```

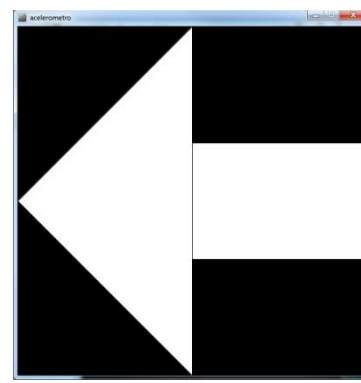
O código acima não usa muito mais conceitos do que os que já vimos até agora. Neste código, recebemos o que vem pela porta serial e salvamos na variável **val**. Como esta variável não é exatamente o número que enviamos pelo Arduino devido a comunicação, temos de converter o valor recebido de decimal para char (vide tabela ASCII). Então, através de uma rotina **switch case**, mostro na tela gerada pelo **Processing** uma seta na posição que o sensor está virado ou uma esfera, caso o sensor esteja na posição normal. Veja abaixo os 5 casos gerados pelo **Processing** com as informações recebidas do Arduino com o acelerômetro:



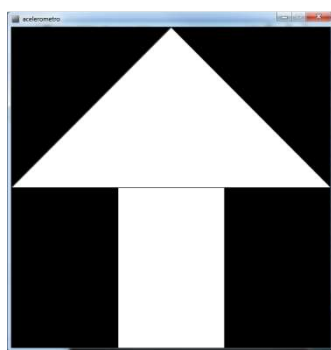
Posição Normal - FLAT



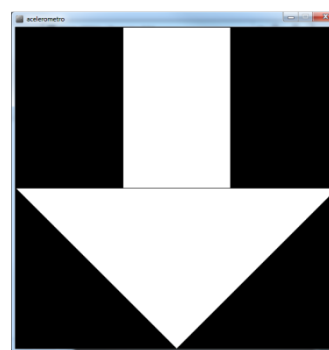
Direita - Landscape Right



Esquerda - Landscape Left



Para Cima - Portrait Up



Para Baixo - Portrait Down

Agora, quando você pegar seu celular ou tablet na mão, e virar para algum dos lados, você já consegue entender um pouco do que está acontecendo ali dentro. Ao escolher um acelerômetro sempre tome como base a tensão de alimentação e comunicação do mesmo, além do tipo de comunicação que ele possui e a resolução. Dominando a leitura e entendimento deste tipo de sensor, é possível fazer diversos tipos de projetos.

Vamos agora aprender a usar outro tipo de sensor, o sensor ultrassônico.

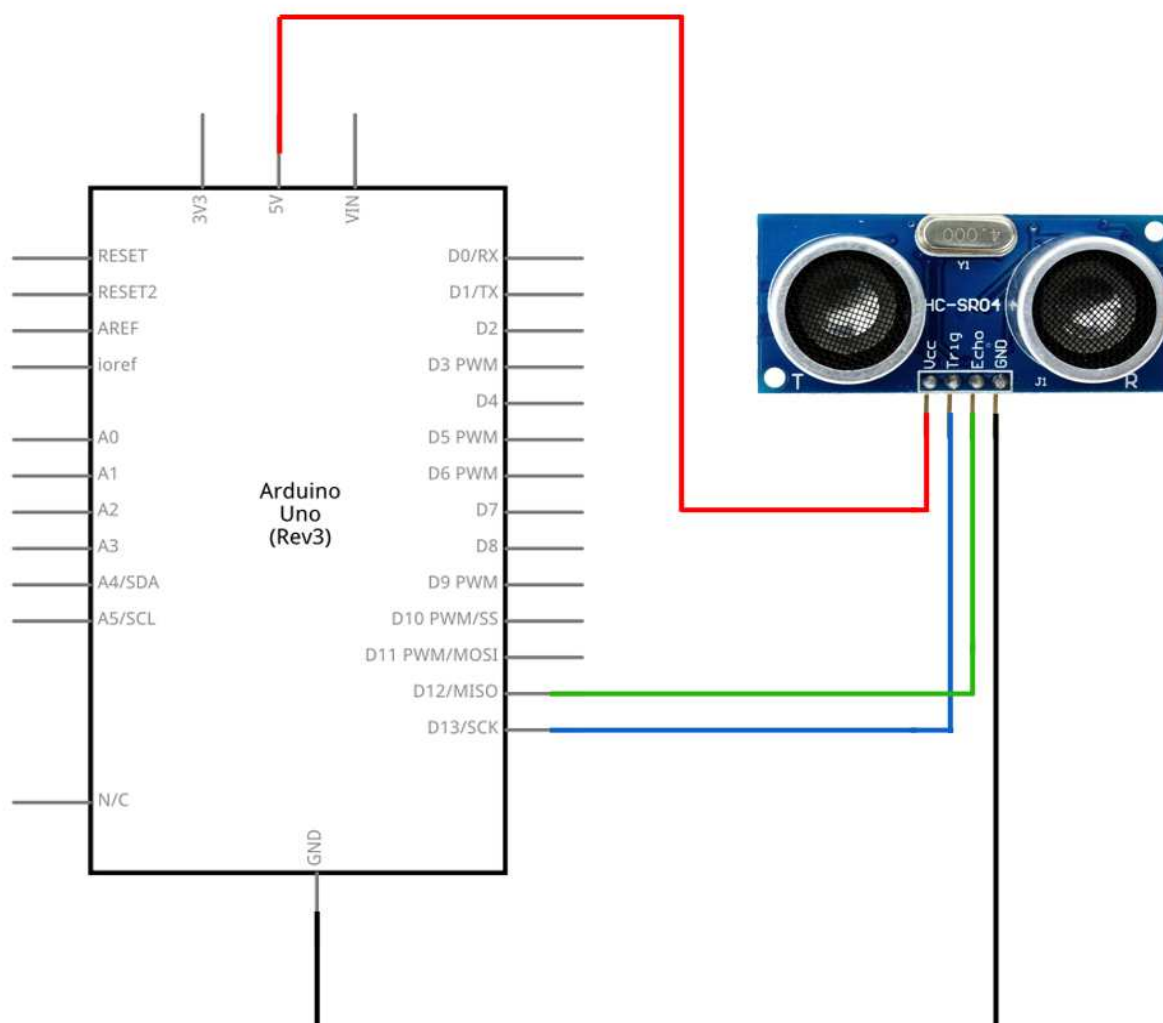
- **Sensor Ultrassônico**

Componentes: 01x Sensor Ultrassônico

Descrição: Aprenda como funciona este sensor, que consegue fazer leituras precisas de distância, com resolução de 1cm.

Sensores ultrassônicos são usados em diversos tipos de aplicações. Basicamente, em qualquer aplicação que você precise saber a distância de um determinado objeto até o sensor em centímetros, é possível usar este tipo de sensor. Hoje são encontrados em robôs de competição, como os robôs de sumô autônomos, que precisam saber a que distância seu oponente está; podem ser usados em ambiente industrial, verificando se existem objetos na esteira de produção; podem ser encontrados em pistolas radares, para saber a velocidade de um corpo; podem ser usados para verificar nível de água em barris; entre muitas outras aplicações. O método de funcionamento deste sensor é simples, ele emite um sinal em uma frequência ultrassônica por um dos pequenos cilindros, e recebe o sinal refletido pelo outro. Através do tempo que demorou para o sinal voltar para o sensor, ele consegue descobrir a distância que o obstáculo está. Isso vai ficar mais fácil de ser entendido ao vermos o código no Arduino. O modelo do sensor que estamos usando é o **HC-SR04**, que consegue fazer leituras de 2cm até 400cm (mais conhecido como 4 metros). Para saber mais sobre o sensor, verifique a página do produto no site da RoboCore.

Vamos começar montando o seguinte circuito:



Agora, grave o seguinte código em seu Arduino:

```
const int trig = 13;
const int echo = 12;

void setup() {
    Serial.begin (9600);
    pinMode(trig, OUTPUT);
    pinMode(echo, INPUT);
}

void loop() {
    long duracao;
    long distancia;
    digitalWrite(trig, LOW);
    delayMicroseconds(2);
    digitalWrite(trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig, LOW);
    duracao = pulseIn(echo, HIGH);
    distancia = duracao / 58;
    Serial.print(distancia);
    Serial.println(" cm");
    delay(500);
}
```

Abra o monitor serial, coloque sua mão na frente do sensor, a aproximadamente uma distância de 5cm e veja na tela o que é informado. Você deve estar vendo algo como:



Caso você tenha uma régua poderá verificar que a precisão deste sensor é muito boa. Mas, como este sensor funciona? Vamos analisar o código que está no Arduino.

Nas duas primeiras linhas do código, informamos que usaremos os pinos digitais 12 e 13. O primeiro será o pino do **trigger** e o segundo será o pino do **echo**. A palavra **trigger** significa gatilho, em inglês. Nesta porta iremos emitir a frequência ultrassônica e no pino **echo** iremos fazer a leitura do sinal recebido, que foi refletido pela barreira. No bloco **setup** do código simplesmente iniciamos a comunicação serial com a taxa de 9600bps e informamos que os pinos trabalharão como saída e entrada, ou seja, para enviar o sinal e para ler o sinal retornado. Como sempre, a mágica acontece no **loop**. Nas duas primeiras linhas dentro do loop criamos duas variáveis do tipo *long*, uma chamada **duracao** e outra **distancia**. Variáveis do tipo *long* são muito parecidas com a do tipo *int*, que estamos acostumados a criar. A grande diferença entre estes dois tipos de variáveis é o tamanho que o número armazenado pode ter. Enquanto que números do tipo *int* podem ir de -32.768 até 32.767, números do tipo *long* podem ir de -2.147.438.648 a 2.147.483.647. Trata-se de uma grande diferença, e usamos isso para ter uma precisão melhor durante a medição. Depois de declarar as duas variáveis do tipo *long*, emitimos o sinal ultrassônico utilizando as funções **digitalWrite**. Primeiro setamos o pino **trigger** como nível lógico baixo durante 2 microsegundos, depois setamos como nível lógico alto (5V) por 10 microsegundos e então voltamos para nível lógico baixo. A única razão de deixarmos o pino em 0V por 2µs no começo é para garantir que ao dar o pulso de 5V, o estado anterior era 0V. Conforme o datasheet do sensor informa, para receber o sinal no sensor é

necessário emitir um sinal no pino *trigger* durante 10µs (vide página 2 do datasheet no site da RoboCore). Quando damos o *trigger*, o sensor envia 8 ciclos de um trem de pulsos com frequência ultrassônica e depois de enviado faremos a leitura do que irá retornar.

Assimilamos, então, à variável *duracao* o resultado do uso da função `pulseIn(echo, HIGH)`. É muito importante agora entendermos o que essa função faz. A função `pulseIn()` lê o tempo de um pulso, seja HIGH ou LOW em um pino. A sintaxe de uso é:

`pulseIn(pino que faremos a leitura, nível lógico esperado);`

Esta função retorna, para a variável que foi atrelada a ela, um período de tempo em microsegundos (lembrando que 1 segundo = 1.000.000 microsegundos, por isso o tipo da variável que irá armazenar é *long*). Funciona da seguinte forma: ao colocarmos HIGH no nível lógico esperado, a rotina começa a contar o tempo quando o pino está em HIGH, até que o pino em questão vá para nível lógico baixo (LOW). Este tempo entre o pino estar alto até ficar baixo é dado em microsegundos e é armazenado, no nosso caso, na variável *duracao*.

Que tempo é este que estamos chamando de *duracao*? Trata-se do tempo em que o sinal rebatido foi recebido, até o tempo que paramos de receber, ou seja, o fim do sinal. Este tempo representa o sinal de ida e volta. Conforme o datasheet, para ter o valor em centímetros a partir deste tempo, basta dividirmos o resultado por **58** (caso quiséssemos o valor em polegadas, bastaria dividir por **148**). Após fazer esta divisão, temos na variável *distancia* o valor já em centímetros da distância entre nosso sensor e o objeto. Então fazemos o *print* do valor da distância seguido da unidade de medida, no caso cm. Agora já sabemos como fazer nossa placa Arduino saber exatamente a quantos centímetros de distância está o objeto a sua frete. Vamos deixar isso um pouco mais gráfico. Usaremos o **Processing** novamente para conversar com nossa placa Arduino e fazer algo a partir da leitura de um sensor.

Para nosso próximo experimento, iremos deixar o Arduino de lado por um instante. O que queremos como resultado final nesta parte da apostila é mostrar um quadrado na tela que irá simular o obstáculo a frente do sensor. Se o sensor estiver muito próximo do obstáculo, o quadrado ficará maior na tela, e se o sensor estiver longe do obstáculo, o quadrado ficará pequeno (como se fosse o que o sensor está "enxergando"). Além disso, vamos querer colocar na tela gráfica a distância escrita. Vamos começar pensando no quadrado que irá andar, no **Processing**. Rode o seguinte código no **Processing**:

```
int janela = 600;
int dimensao = 0;

void setup() {
  size(janela, janela); //criamos janela quadrada, de tamanho 600
  rectMode(CENTER);
}

void draw() {
  dimensao = dimensao + 1; //poderia ser substituido simplesmente por dimensao++;
  if (dimensao >= janela) { //garantimos que o tamanho máximo do quadrado seja o tamanho da
    janela
    dimensao = 0;
  }
  clear(); //limpa a tela para cada novo retângulo desenhado
  rect(janela/2, janela/2, dimensao, dimensao); //desenha o novo quadrado
  //seu centro será sempre no meio da janela, por isso as coordenadas X e Y são janela/2
  //seu tamanho será sempre igual a variável dimensao
}
```

O intuito desse código é simplesmente desenhar um quadrado no centro da tela e fazer "andar", ou seja, aumentar de tamanho gradativamente parecendo que está querendo sair da tela. Quando ficar no maior tamanho possível, recomeçamos com o quadrado pequeno. A primeira coisa que fazemos é declarar a variável *janela* e a variável *dimensao*. A variável *janela* representa o tamanho da janela que iremos criar ao iniciar o programa. Já a variável *dimensao* vai servir para sabermos o tamanho do quadrado enquanto o desenhamos. No **setup** do programa, criamos a janela propriamente dita usando o comando **size** e informamos ao programa que iremos usar o tipo

de retângulo *CENTER*. Isso quer dizer que ao desenhar o quadrado (que nada mais é que um retângulo de lados iguais), iremos tomar sempre como referência o centor do mesmo para posicionamento e tamanho de arestas. Na seção **draw** do programa, começamos incrementando a dimensão das arestas do quadrado em 1 unidade. Veja que esta linha poderia ser substituída por *dimensao++*.

Logo após incrementarmos a variável que dá o tamanho do quadrado, verificamos se o tamanho do mesmo é maior ou igual ao da nossa janela, ou seja, se for maior nosso quadrado está no tamanho máximo e não é necessário crescer mais, então voltamos com o valor de dimensão para **zero**. Caso o tamanho das arestas não seja maior ou igual ao tamanho da janela, pulamos a instrução de igualar dimensao a zero. Depois dessa verificação, limpamos a tela, para que a janela não fique com nenhum resquício de quadrado antigo (se não limparmos a janela, o programa vai desenhar vários quadrados, um dentro do outro). E por fim, desenhamos o quadrado propriamente dito. Para desenhar o quadrado usamos o seguinte comando:

```
rect(posição de origem em x, posição de origem em y,  
      tamanho da aresta em x,tamanho da aresta em y);
```

Conforme nosso código, a posição de origem em x e y sempre será o tamanho da janela dividido por 2. Como a janela é constante e não vai mudar durante o programa, este valor sempre será o mesmo. Se o tamanho da janela for de 600x600 pixels, a posição central do quadrado **sempre** vai ser 300 em x e 300 em y. Então usamos para tamanho da aresta em x e em y o mesmo valor, ou seja, **dimensao**. Como queremos desenhar um quadrado, as duas arestas tem sempre que ser iguais. Clique no botão **RUN** do **Processing** e veja o quadrado pequeno ir aumentando até querer sair da tela.

Agora que temos a parte gráfica quase pronta, temos que gravar um novo código no Arduino que irá enviar as informações que precisamos pelo cabo USB até o computador, onde o **Processing** vai recebê-los e tratá-los. Grave o seguinte código no Arduino:

```
const int trig = 13;  
const int echo = 12;  
  
void setup() {  
  Serial.begin (9600);  
  pinMode(trig, OUTPUT);  
  pinMode(echo, INPUT);  
}  
  
void loop() {  
  long duracao;  
  long distancia;  
  digitalWrite(trig, LOW);  
  delayMicroseconds(2);  
  digitalWrite(trig, HIGH);  
  delayMicroseconds(10);  
  digitalWrite(trig, LOW);  
  duracao = pulseIn(echo, HIGH);  
  distancia = duracao / 58;  
  if(distancia >= 20){  
    Serial.print("a");  
  }else{  
    Serial.write(distancia);  
  }  
  delay(50);  
}
```

Este código é muito semelhante ao que usamos há pouco no Arduino, porém agora usamos o comando **Serial.write** para enviar os números da distância. Usamos o **write** apenas para não precisar fazer a conversão conforme a tabela ASCII. Também enviamos outra coisa pelo cabo USB além do valor da distância, você sabe dizer o que é e quando é enviado? Nós iremos fazer uma pequena limitação no código e fazer com que, caso a distância até o obstáculo seja maior ou igual a

20cm, enviaremos uma letra **a**. No **Processing**, iremos fazer um tratamento tal que, caso recebamos essa letra, saberemos que estamos fora do *range* de distância permitido.

Agora, coloque no **Processing** o seguinte código:

```
import processing.serial.*;
Serial port;
int val;

int janela = 600;
int dimensao = 0;

void setup() {
  println(Serial.list());
  port = new Serial(this, Serial.list()[0], 9600);
  size(janela, janela); //criamos janela quadrada, de tamanho 600
  rectMode(CENTER);
  textFont(createFont("Arial", 16));
}

void draw() {
  background(0);

  if (port.available() > 0) {
    val = port.read(); //recebo a variavel em decimal que representa um char conforme ASCII table
  }
  if (val == 'a') {
    clear();
    textSize(26);
    textAlign(CENTER);
    text("COLOQUE UM OBJETO A MENOS DE 20 CM DE DISTÂNCIA DO SENSOR", janela/2,
janela/2, janela, 100);
  } else
  {
    clear();
    textSize(26);
    textAlign(CENTER);
    text("Distância: " + val + " cm", janela/2, janela);
    rect(janela/2, janela/2, (-janela*0.05*val + janela), (-janela*0.05*val + janela));
    //desenha o novo quadrado
  }
}
```

Clique no botão **RUN** e brinque com um objeto na frente do sensor. A ideia é que você consiga visualizar graficamente o que o sensor está "enxergando" em sua frente. O código no **Processing** não mudou muito, apenas colocamos os comandos básicos para ele aceitar comunicação com o Arduino e no **draw** fazemos a leitura do que chega pela porta serial. Se o que chegar for aquela letra **a**, escrevemos no centro da tela a frase:

COLOQUE UM OBJETO A MENOS DE 20 CM DE DISTÂNCIA DO SENSOR

Caso contrário, ou seja, caso eu tenha um objeto a uma distância menor que 20cm, eu desenho o quadrado na tela conforme o valor que recebo pela serial. A função que indica o tamanho do quadrado em x e y pode parecer estranha, mas usamos ela unicamente para que quando o objeto estiver perto, ou seja, com um valor de distância pequeno, o quadrado seja grande e vice-versa.

/***** THE END *****/

Aqui foram colocados apenas códigos introdutórios para componentes de nível avançado. Como as opções de códigos e projetos são muito amplas, não é possível contemplar todas em um único manual, porém para qualquer dúvida nestes projetos, sinta-se a vontade em entrar em contato com a RoboCore pelo e-mail info@robocore.net

Arduinize o Mundo
RoboCore - Kit Avançado para Arduino
WWW.ROBOCORE.NET