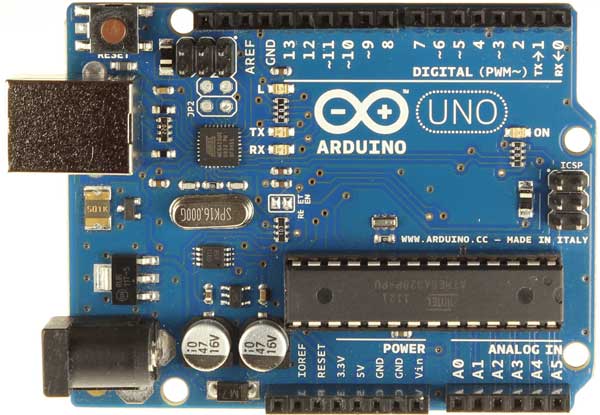
Ao pesquisar [***MeArm***](https://www.google.com.br/search?q=MeArm&safe=active&source=lnms&tbm=isch&sa=X&ved=0ahUKEwj096m8_K7TAhXDPpAKHR8eBDsQ_AUICigD&biw=1278&bih=699) na internet, temos uma ideia de como ficará o projeto. Além do braço-robô, teremos dois joysticks para controlá-lo.

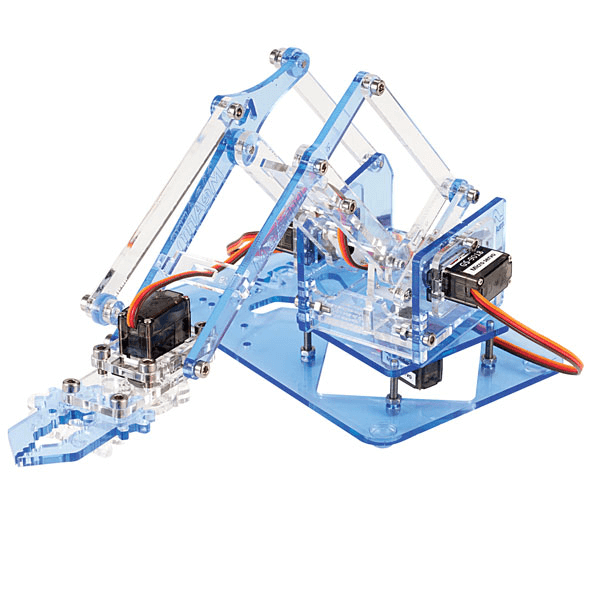
Neste vídeo, vamos detalhar os materiais necessários para a montagem do nosso **braço-robô**. Vamos lá:

* **Arduino**

****

No treinamento, utilizaremos o **Arduino UNO**, mas não se preocupe se você não a possui, os outros tipos também funcionam perfeitamente para programar a inteligência do braço-robô.

* **Kit de Chassi Braço Robótico**

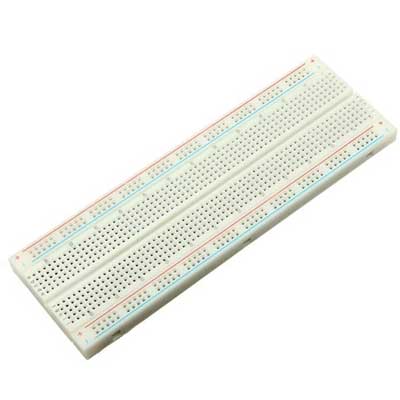
****

O chassi utilizado no curso é de **acrílico**, mas o projeto funciona também no chassi de **MDF**. O chassi de acrílico foi escolhido por conta da sua resistência e durabilidade.

[Link do Mercado Livre](http://lista.mercadolivre.com.br/kit-bra%C3%A7o-arduino#D[A:kit-bra%C3%A7o-arduino,L:1])

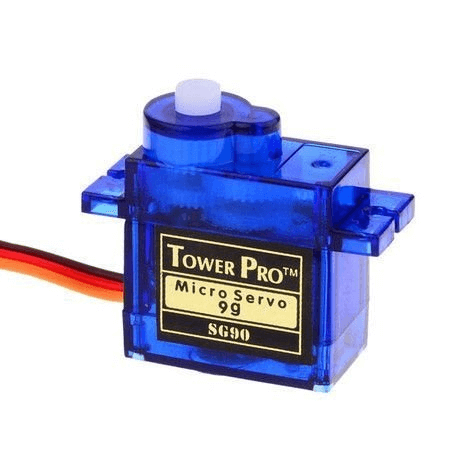
*Atenção: Existem kits que já vem com os motores servo!*

* **Protoboard**

****

A protoboard será utilizada basicamente para a alimentação dos motores, então ela pode ser tanto a grande quanto a mini (menor).

* **Quatro Servo Motores**

****

Um servo motor bem popular, fácil de ser encontrado na internet e barato, é o **Micro Servo Motor 9g SG90**. Existem outros servo motores mais sofisticados, mas esse atende muito bem ao nosso projeto.

[Link no Mercado Livro](http://lista.mercadolivre.com.br/micro-servo-motor-9g#D[A:micro-servo-motor-9g-])

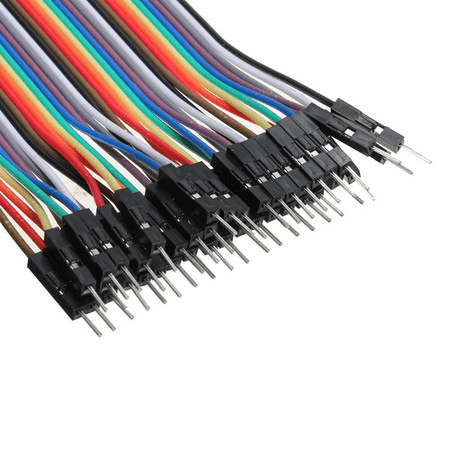
* **Dois Joysticks Arduino 3 Eixos**

****

[Link no Mercado Livro](http://lista.mercadolivre.com.br/m%C3%B3dulo-joystick-arduino#D[A:m%C3%B3dulo-joystick-arduino])

Precisaremos de dois Joysticks Arduino 3 Eixos, que utilizaremos para controlar o braço-robô.

* **Jumpers - 8 Macho x Fêmea e 14 Macho x Macho**

****

Utilizaremos oito *jumpers* Macho x Fêmea e catorze Macho x Macho, para fazer as ligações da Protoboard com o chassi e afins.

**Lista resumida**

* **Arduino**
* **Kit de Chassi Braço Robótico**
  + *um pouco de paciência para montar o braço :)*
* **Protoboard**
* **Quatro Servo Motores**
* **Dois Joysticks Arduino 3 Eixos**
* **Jumpers - 8 Macho x Fêmea e 14 Macho x Macho**

Com isso nossa lista está pronta. Todos esses itens podem ser achados no Mercado Livre ou até mesmo em lojas de eletrônica.

Para que possamos interagir mais com o Arduino, nesse vídeo faremos um teste com o **servo motor**. Ligaremos-o ao Arduino, para começar a entender o seu funcionamento.

**Detalhando a fiação do Servo Motor 9g**

Para ligar o servo motor ao Arduino, precisaremos de **três *jumpers* macho x macho**. A fiação dele possui três cores, **laranja**, **vermelho** e **marrom**:



Cada fio possui uma representação:

* **Laranja**: fio de controle do motor, que é conectado a um pino PWM do Arduino (veremos isso mais à frente)
* **Vermelho**: fio de alimentação
* **Marrom**: fio terra (GND)

Para não ligar o servo motor a uma protoboard, e da protoboard conectamos ao Arduino, utilizaremos os *jumpers* para realizar a conexão direta ao Arduino.

**Conectando o Servo Motor 9g ao Arduino**

Para interagir com o motor, executaremos um código que **já está pronto**. Basta abrir a IDE do Arduino, e no menu superior, selecionar ***File -> Examples -> Servo -> Sweep***.

Vamos nos atentar à função **setup()**:

void setup() {   
  myservo.attach(9);  
}

O programa utiliza o **pino 9**. Logo, vamos ligar o fio laranja do servo motor a um *jumper*, que por sua vez é ligado ao **pino 9** do Arduino. Por último, ligamos um *jumper* ao fio vermelho e um *jumper* ao fio preto, ambos do servo motor, que serão ligados aos pinos **5V** e **GND** do Arduino, respectivamente.

**Executando o programa e testando o motor**

Agora, podemos ligar a alimentação do Arduino e conectá-lo à nossa máquina, para executar o programa.

Ainda na IDE do Arduino, subimos o código para a placa clicando no botão de ***Upload***(uma seta para a direita).

O servo motor começa a girar. Se formor prestar atenção no código, vemos que o código faz com que o motor fique girando de 0º até 180º, e depois de 180º até 0º.

Isso faz com que o motor fique indo e voltando, o que nos ajuda a perceber o funcionamento do servo motor, já que o seu **giro é de 180º**, ele não gira 360º.

Então, com esse código, podemos testar todos os nossos servo motores, para verificar que eles estão em pleno funcionamento.

Sobre o servo motor é correto afirmar:

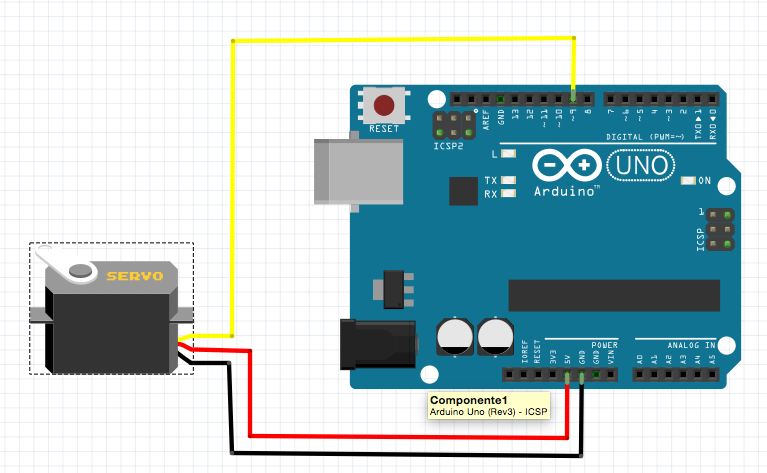
1. O servo motor é uma máquina eletromecânica, que apresenta movimento proporcional a um comando.
2. A maior parte dos servo motores, diferentes dos motores contínuos, giram 360 graus.
3. Através de um sinal de controle é possível mudar a posição atual para uma desejada.

Todas as afirmativas são verdadeiras, exceto a afirmativa 2, pois em geral os servos motores trabalham com ângulos que variam de 0 a 180 graus. O servo motor é um dispositivo, eletromecânico, que apresenta movimento proporcional a um comando, ou seja, recebe um valor de entrada e atribui um ângulo como saída. Em contraste com os motores contínuos, que giram indefinidamente, o eixo dos servo motores trabalham com ângulos de 0 de 180º graus (mas alguns modelos podem giram 360 graus).

Você já viu no video como conectar um *Servo Motor* na placa Arduino mas agora é a hora de testar os seus motores. Para tal:

1) Ligue cada motor na placa do Arduino. Use o pino 9 para o controle.

Segue o sketch do Fritzing com o cicuito:



*Lembre-se das cores*:

* Laranja/Amarelo representa o pin de controle (aqui PIN9)
* Vermelho é a alimentação (5V)
* Marrom serve para a terra (GND)

Observação: Você também pode baixar o sketch do Fritzing [AQUI](https://s3.amazonaws.com/caelum-online-public/arduino-robotica/sketches/servo-simples.fzz).

2) Teste o código abaixo na Arduino IDE:

#include <Servo.h>  
  
Servo myservo;  // cria o objeto servo  
int pos = 0;    // a variavel que representa a posicao do servo  
  
void setup() {  
  myservo.attach(9);    
}  
  
void loop() {  
  for (pos = 0; pos <= 180; pos += 1) { // de 0 a 180 graus  
    myservo.write(pos);              // "escrevendo" para o motor a posicao atual  
    delay(15);                       // espereando 15ms  
  }  
  for (pos = 180; pos >= 0; pos -= 1) { // de 180 até 0 graus degrees  
    myservo.write(pos);               
    delay(15);                         
  }  
}

3) Envie o código para a placa Arduino e verifique o Servo!

Neste capítulo, aprenderemos mais sobre o **servo motor**, e mais importante ainda, sobre o **PWM** (***P****ulse* ***W****idth* ***M****odulation*), um recurso que é muito utilizado na robótica.

**Conhecendo o Servo Motor 9g**

Escolhemos o **Micro Servo Motor 9g SG90** para o nosso projeto, que é barato e fácil de ser encontrado. Justamente por ser mais barato, ele possui as engrenagens de plástico, mas que nos atendem perfeitamente.

Mas afinal, o que é um servo motor? Os motores, em eletrônica, são chamados de **atuadores**, que nada mais são do que dispositivos que convertem energia em movimento

Sempre que estamos trabalhando com novos componentes, é válido consultar a sua documentação. Podemos consultar a documentação do Micro Servo Motor 9g SG90 [aqui](http://micropik.com/PDF/SG90Servo.pdf).

Nessa documentação, conseguimos saber todos as especificações sobre o servo motor, como o seu torque, sua tensão de operação, temperatura e até suas dimensões.

Sempre que for usar um componente novo consultar na internet nome + datasheet para ver as especificações do fabricante

Agora que já conhecemos melhor sobre o **servo motor**, vamos entender sobre o **PWM**(***P***ulse ***W***idth ***M***odulation).

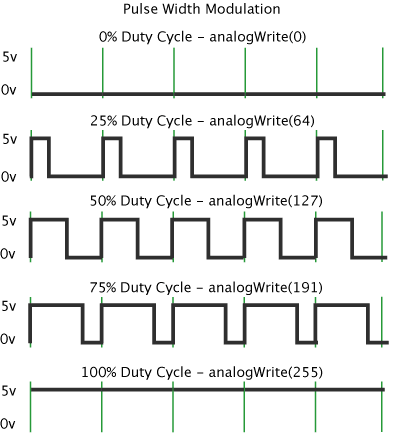
## Entendendo o PWM

Dentro do nosso projeto, o servo motor será colocado em ângulos específicos, através de sinais de controle que iremos colocar nele. Mas como faremos isso?

Utilizaremos o **PWM** (***P***ulse ***W***idth ***M***odulation, ou Modulação de Largura de Pulso), então vamos entender como ele funciona.

No mundo digital, temos o **LOW**, que é o zero, e o **HIGH**, que no nosso caso é o 5 volts (se estivéssemos trabalhando com o Raspberry Pi, seria 3.3 volts, por exemplo), que podemos ser interpretados como **0** (**LOW**) e **1** (**HIGH**).

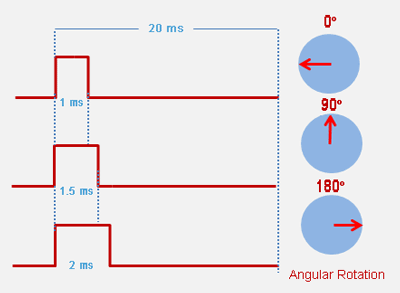
Se temos 0 e 1, como iremos traduzir isso para ângulos diferentes do motor? A resposta é que não tem como! É aí que entra o **PWM**:



A técnica consiste em variar, dentro de um período de tempo (na imagem, os períodos estão divididos pelas colunas verdes), o tempo em que o pulso se mantém em **HIGH**. Ou seja, através da largura do pulso de uma onda quadrada, é possível o controle de potência ou velocidade.

## Aplicando o PWM no servo motor

Ao olhar a especificação do nosso motor, vemos que podemos mandar um pulso de 1 a 2 milissegundos. Ao mandar um pulso de 1 milissegundo, a motor fica posicionado em 0°, se mandarmos um pulso de 1.5 milissegundos, ele fica posicionado em 90°, e se o pulso for de 2 milissegundos, o motor fica posicionado em 180°:

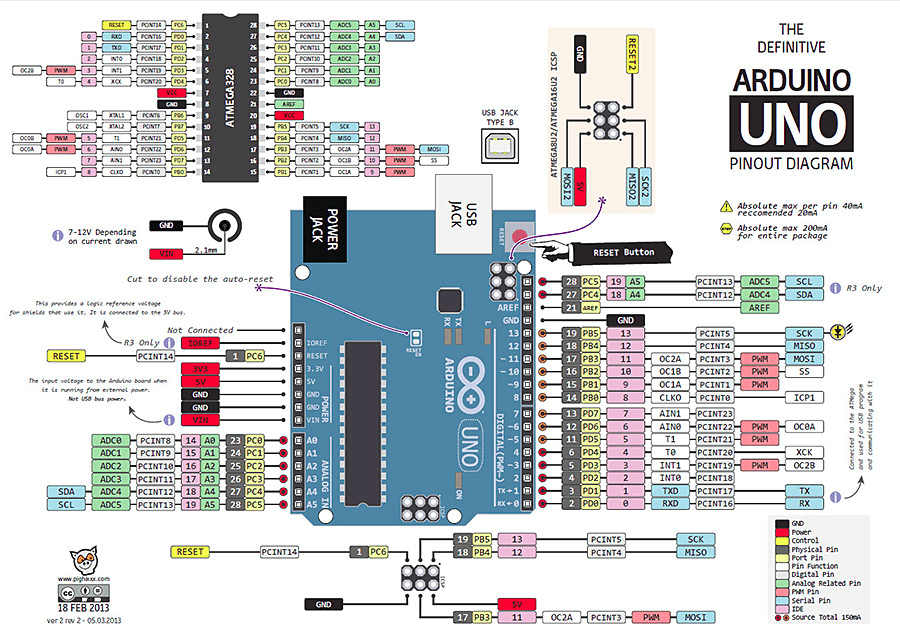


Logo, para cada valor do pulso, entre 1 e 2 milissegundos, o motor ficará em uma angulação diferente. Por exemplo, para posicionar o motor em 45°, mandamos um pulso de 1.25 milissegundos.

Essa variação do tempo, no qual colocamos o sinal em HIGH, será traduzida (no caso do servo motor) no ângulo em que o servo motor ficará posicionado. Mas se queremos o motor em 37°, por exemplo, precisamos aplicar uma regra de três para saber a duração do tempo que devemos mandar? Felizmente não, pois a biblioteca do servo motor no Arduino, quando colocamos o ângulo que queremos trabalhar, já sabe o tempo em que irá trabalhar. Logo, só precisamos escrever o ângulo que quisermos.

## Relação de pinos PWM do Arduino

Por fim, é importante sabermos quais e quantos pinos PWM o Arduino possui:



Ele possui 6 pinos PWM, são eles: **3**, **5**, **6**, **9**, **10** e **11**. E são exatamente esses pinos que utilizaremos para controlar o nosso motor.

Sobre os estados das entradas digitais, é correto afirmar que:

## Existem somente dois estados.

## Também podem receber valores intermediários, que variam entre o LOW e o HIGH.

## Por convenção, no Arduino entendemos que HIGH representa ****5V**** e LOW ****0V****.

## Todas estão incorretas.

As afirmativas 1 e 3 estão corretas. As entradas digitais só podem assumir dois estados, **HIGH** e **LOW**, ou seja, **0V** ou **5V**. Para que possamos ter valores intermediários, é necessário utilizarmos as entradas **analógicas**.

No capítulo passado, utilizamos um código de exemplo da própria IDE do Arduino, para interagirmos com o servo motor. Vamos aproveitar esse código, mas fixando alguns ângulos, para entendermos como faremos o braço-robô funcionar.

## Colocando o motor em determinados ângulos

Lembrando que o código de exemplo da IDE do Arduino pode ser visto no menu superior, selecionando **File -> Examples -> Servo -> Sweep**:

#include <Servo.h>

Servo myservo;

int pos = 0;

void setup() {

myservo.attach(9);

}

void loop() {

for(pos = 0; pos < 180; pos += 1) {

myservo.write(pos);

delay(15);

}

for(pos = 180; pos>=1; pos-=1) {

myservo.write(pos);

delay(15);

}

}

Ao invés de fazer as alterações dentro da função **loop()**, vamos fazer dentro da função **setup()**. Então vamos apagar todo o seu conteúdo:

#include <Servo.h>

Servo myservo;

int pos = 0;

void setup() {

myservo.attach(9);

}

void loop() {

}

Há uma variável **pos**, que representa o ângulo em que o servo motor ficará posicionado e novamente o servo motor deve ficar no **pino 9**, que é um pino PWM do Arduino. Para escrever, dizer qual ângulo queremos, utilizamos a função **write**:

#include <Servo.h>

Servo myservo;

int pos = 0;

void setup() {

myservo.attach(9);

myservo.write(pos);

}

void loop() {

}

Ou seja, o servo ficará posicionado em 0°, valor da variável **pos**. Se modificarmos o seu valor para 90, o servo motor ficará posicionado em 90°. Logo, é com essa função que conseguimos posicionar o motor no ângulo exato que quisermos.

Vamos colocar o motor em 90°, assim conseguimos posicionar melhor o suporte. Basta executar o programa, remover o suporte e reposicioná-lo. Ele deve ficar assim:



Assim fica mais fácil identificarmos o ângulo do motor. **É importante fazer isso para todos os motores**.

Através desta biblioteca (**Servo.h**), temos como posicionar o servo motor no ângulo desejado. De forma resumida, sua função está em converter os valores de pulso para os ângulos. Lembrando ainda que ela já vem instalada por padrão na IDE do Arduino.

Para importar a biblioteca devemos usar o comando #include

#include <Servo.h> //importando a biblioteca

E para usar o servo é preciso declarar a variável do tipo Servo:

Servo motor; // cria um objeto que representa o motor

A partir daí podemos trabalhar com o objeto motor.

Segue também a documentação da biblioteca no site do Arduino: [LINK](https://www.arduino.cc/en/Reference/Servo)

Neste exercício vamos alinhar todos os motores para que eles fiquem com o seu suporte na mesma orientação.

1- O primeiro passo é carregarmos no nosso Arduino o código que coloca os motores a 90 graus. Em sua Arduino IDE, faça o código abaixo:

#include <Servo.h>

Servo myservo;

int pos = 90;

void setup() {

myservo.attach(9);

myservo.write(pos);

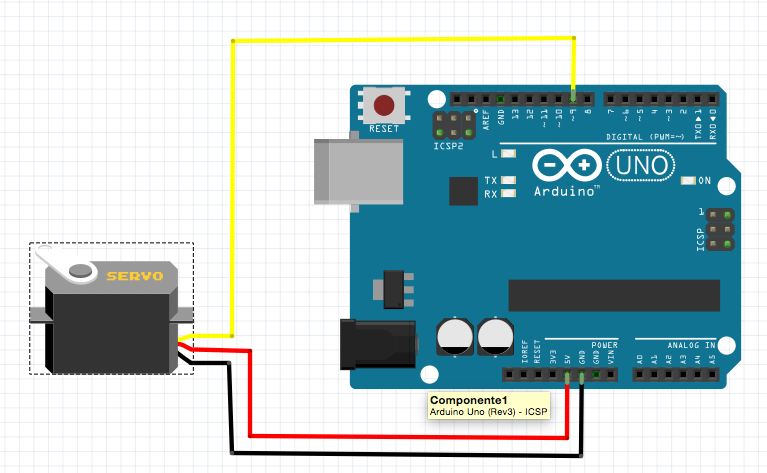
}

void loop() {

}

Este código pega o servo que está no pino 9 e o movimenta para 90 graus.

2- Agora que já temos o código, ligue o servo ao seu Arduino como na imagem abaixo:



3- Ligue o cabo USB ao seu computador e ao Arduino e transfira o código. Você verá que assim que o código for executado o braço do seu servo moverá para uma posição fixa, que é a posição de 90 graus dele.

4- Agora você deve remover o braço/suporte do seu servo e posicioná-lo como abaixo:

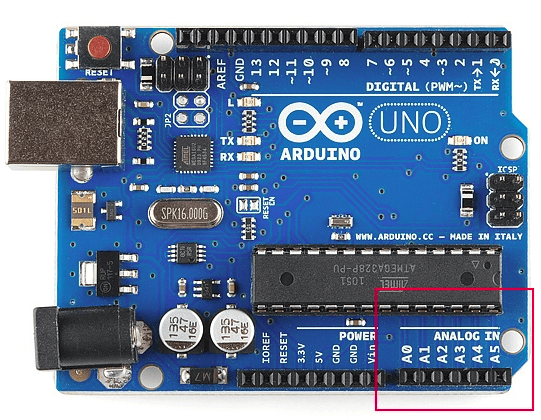


Deste modo, garantimos que o suporte esteja acompanhando a direção do corpo do servo, o que facilitará a nossa montagem e programação do braço robótico.

5- Agora você deve repetir este processo com **todos** os seus servos, para que todos estejam seguindo um mesmo padrão e fiquem alinhados a 90 graus seguindo o corpo do servo.

Lembre-se, execute o código, deixe o servo achar a sua posição de 90 graus e só em seguida altere a posição do suporte para que fique alinhado ao servo!

Agora que conhecemos melhor as portas digitais, e principalmente as PWM, do Arduino, vamos focar nesse vídeos nas suas **portas analógicas** (as portas dento do retângulo vermelho, na imagem abaixo). A parte analógica do projeto é tão importante quanto a parte digital.



## Valores das portas analógicas

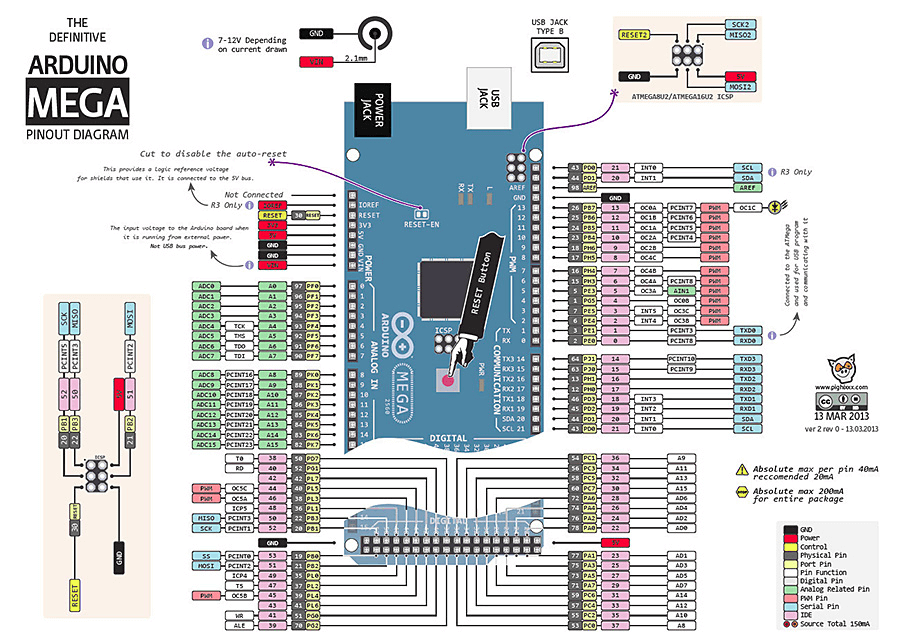
Sabemos que na eletrônica digital, temos o **0** e **1**, que representam o **LOW** e o **HIGH**, especificamente. Com somente esses dois estados, como conseguimos ter uma variação na nossa informação de entrada?

Quando trabalhamos com as portas analógicas, diferentemente de **0** ou **1**, o Arduino possui internamente um conversor analógico-digital, que vai traduzir a entrada em um intervalo que vai de 0 até 1023, ou seja, uma variação de valores muito maior.

Faremos o mapeamento desses valores para a saída que quisermos trabalhar. Isso ficará mais claro quando fizermos o mapeamento do joystick para o servo motor, mais à frente.

## Relação de portas analógicas do Arduino

Podemos ver na imagem acima que o Arduino UNO possui 6 portas analógicas. Mas e se precisarmos de mais? Nesse caso, não podemos utilizar o Arduino UNO, mas podemos utilizar o **Arduino MEGA**:

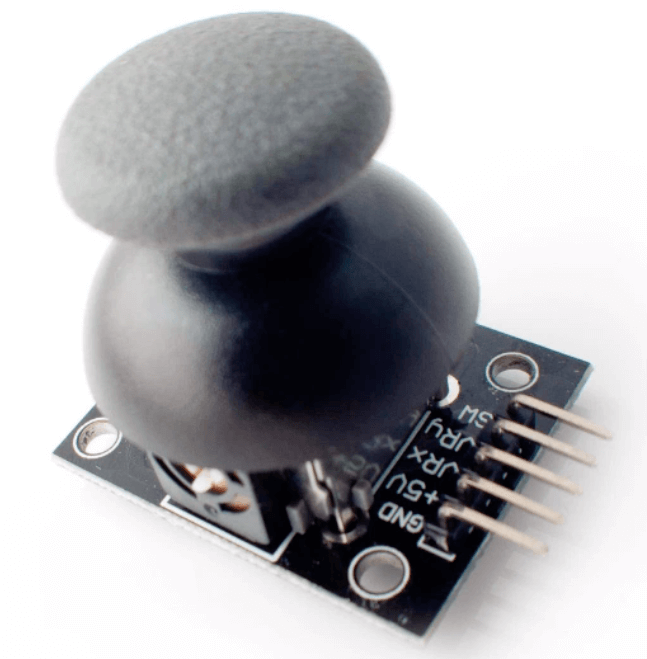


Essa placa possui **16 portas analógicas**, aumentando o leque de portas para diferentes projetos.

Agora que já conhecemos melhor o PWM e as entradas analógicas, vamos reunir esse conhecimento para colocar em prática a montagem e a programação do braço-robô.

**Conhecendo o Módulo Joystick Arduino 3 Eixos**

Começaremos pelo módulo do joystick:



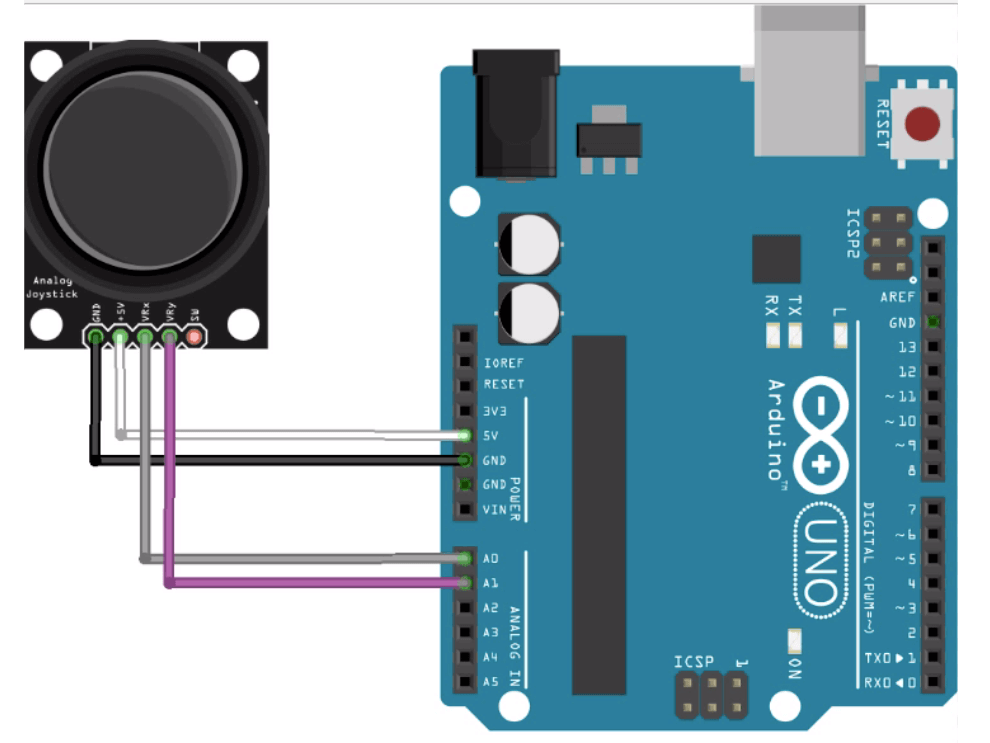
A sua pinagem consiste em:

* **GND**: Terra
* **+5V**: 5 volts, alimentação
* **VRx**: Eixo X do joystick
* **VRy**: Eixo Y do joystick
* **SW**: *Switch*, quando o botão é pressionado

No nosso projeto, não vamos utilizar o pino **SW**, mas é mais uma possibilidade que o módulo nos dá.

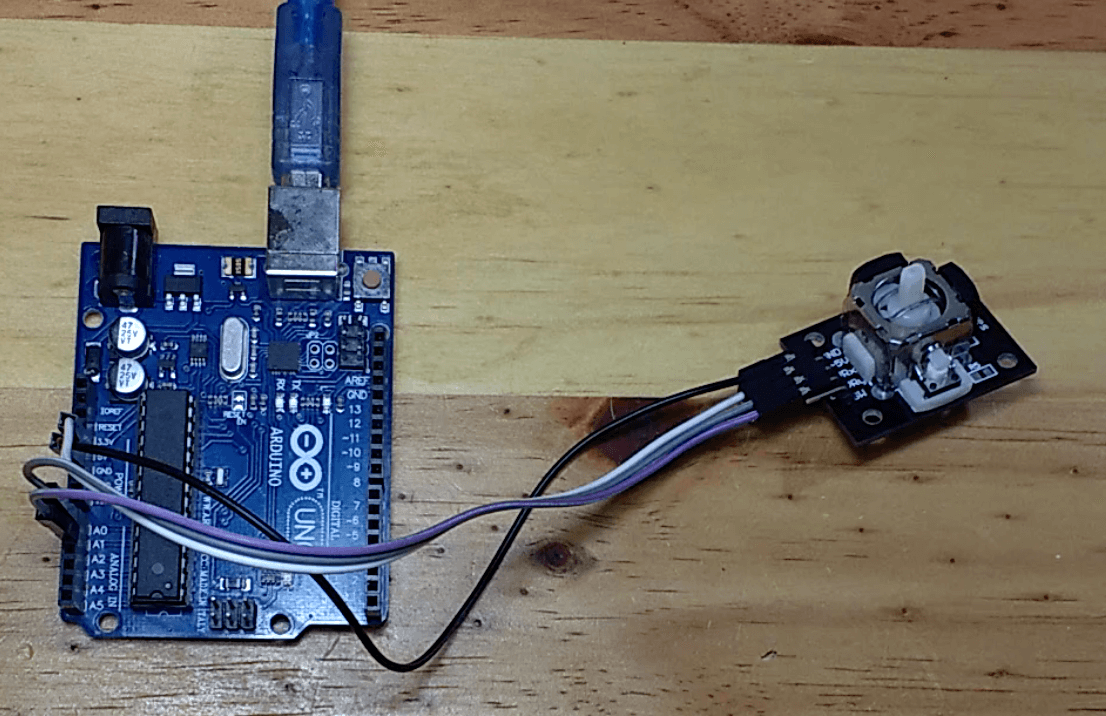
**Desenhando e conectando o joystick ao Arduino**

Para melhorar a visualização, podemos desenhar o projeto no Fritzing:



Assim conseguimos esquematizar e documentar melhor o projeto. Para testar o joystick, vamos ligá-lo diretamente ao Arduino.

Então, baseado no desenho, podemos conectar o módulo do joystick ao Arduino:



Ao conectar e ligar tudo, vamos programar e testar o joystick no próximo vídeo.

Na IDE do Arduino, vamos começar a fazer a programação do joystick, para podermos interagir com ele.

## Inicialização do joystick

Vamos começar definindo constantes, fazendo o mapeamento dos eixos X e Y do joystick, dizendo em que portas eles se encontram. O X se encontra na porta analógica 0 (**A0**) e o Y na porta analógica 1 (**A1**):

#define joystick1X A0

#define joystick1Y A1

void setup() {

}

void loop() {

}

Como dados serão enviador para essas portas, precisamos configurá-las como entrada, no caso, **INPUT**:

#define joystick1X A0

#define joystick1Y A1

void setup() {

pinMode(joystick1X, INPUT);

pinMode(joystick1Y, INPUT);

}

void loop() {

}

Para visualizar como o joystick está funcionando, vamos utilizar o terminal Serial. Então vamos inicializá-lo:

#define joystick1X A0

#define joystick1Y A1

void setup() {

pinMode(joystick1X, INPUT);

pinMode(joystick1Y, INPUT);

Serial.begin(9600)

}

void loop() {

}

Agora, no loop, vamos ler as portas analógicas e imprimi-las.

## Lendo as portas analógicas

O Arduino irá receber informações nas portas analógicas **A0** e **A1**, então precisamos lê-las, através da função **analogRead**. O valor da leitura das portas serão guardadas em duas variáveis:

#define joystick1X A0

#define joystick1Y A1

void setup() {

pinMode(joystick1X, INPUT);

pinMode(joystick1Y, INPUT);

Serial.begin(9600);

}

void loop() {

int x, y;

x = analogRead(joystick1X);

y = analogRead(joystick1Y);

}

Agora basta imprimir o valor dessas duas variáveis na tela:

#define joystick1X A0

#define joystick1Y A1

void setup() {

pinMode(joystick1X, INPUT);

pinMode(joystick1Y, INPUT);

Serial.begin(9600);

}

void loop() {

int x, y;

x = analogRead(joystick1X);

y = analogRead(joystick1Y);

Serial.print(x);

Serial.print(" , ");

Serial.print(y);

}

Para garantir um intervalo entre as leituras, vamos adicionar um delay no programa de 100 milissegundos:

#define joystick1X A0

#define joystick1Y A1

void setup() {

pinMode(joystick1X, INPUT);

pinMode(joystick1Y, INPUT);

Serial.begin(9600);

}

void loop() {

int x, y;

x = analogRead(joystick1X);

y = analogRead(joystick1Y);

Serial.print(x);

Serial.print(" , ");

Serial.print(y);

delay(100);

}

Com o código implementado, vamos testá-lo no próximo vídeo.

Com o programa pronto, podemos salvá-lo e conectar o Arduino ao nosso computador, para testar o joystick.

Podemos fazer o upload do programa para o Arduino. Vemos uma saída parecida com a seguinte:

516 , 513

516 , 513

516 , 513

516 , 513

516 , 513

...

Qual a explicação para esses valores estarem sendo impressos? Lembra que nas portas analógicas, conseguiríamos ter valores entre 0 e 1023? Esses valores são mais ou menos a metade de 1024! Já que o eixo X e o eixo Y do joystick ficam posicionados no meio.

Ao colocar o eixo X do joystick todo para a esquerda, temos o valor **0** sendo impresso, e se o colocarmos todo para a direita, vemos o valor **1023** na impressão. A mesma coisa para o eixo Y, se ele for colocado todo para cima, vemos o valor **0** no terminal, e se ele for colocado todo para baixo, vemos o valor **1023** sendo impresso.

Conforme X e Y forem oscilando, vamos mapeando esses valores diretamente para o servo motor, assim o ângulo do motor vai sendo alterado conforme o valor de X e Y.

Veremos isso nos próximos vídeos.

Chegou a hora de implementar o circuito apresentado neste capítulo!

1 - Vamos começar definindo constantes, fazendo o mapeamento dos eixos X e Y do joystick, dizendo em que portas eles se encontram. O X se encontra na porta analógica 0 (**A0**) e o Y na porta analógica 1 (**A1**):

#define joystick1X A0

#define joystick1Y A1

void setup() {

}

void loop() {

}

Como dados serão enviador para essas portas, precisamos configurá-las como entrada, no caso, **INPUT**:

#define joystick1X A0

#define joystick1Y A1

void setup() {

pinMode(joystick1X, INPUT);

pinMode(joystick1Y, INPUT);

}

void loop() {

}

Para visualizar como o joystick está funcionando, vamos utilizar o terminal Serial. Então vamos inicializá-lo:

#define joystick1X A0

#define joystick1Y A1

void setup() {

pinMode(joystick1X, INPUT);

pinMode(joystick1Y, INPUT);

Serial.begin(9600)

}

void loop() {

}

Agora, no *loop*, vamos ler as portas analógicas e imprimi-las.

2 - O Arduino irá receber informações nas portas analógicas **A0** e **A1**, então precisamos lê-las, através da função **analogRead**. O valor da leitura das portas serão guardadas em duas variáveis:

#define joystick1X A0

#define joystick1Y A1

void setup() {

pinMode(joystick1X, INPUT);

pinMode(joystick1Y, INPUT);

Serial.begin(9600);

}

void loop() {

int x, y;

x = analogRead(joystick1X);

y = analogRead(joystick1Y);

}

Agora basta imprimir o valor dessas duas variáveis na tela:

#define joystick1X A0

#define joystick1Y A1

void setup() {

pinMode(joystick1X, INPUT);

pinMode(joystick1Y, INPUT);

Serial.begin(9600);

}

void loop() {

int x, y;

x = analogRead(joystick1X);

y = analogRead(joystick1Y);

Serial.print(x);

Serial.print(" , ");

Serial.print(y);

}

3 - Para garantir um intervalo entre as leituras, vamos adicionar um *delay* no programa de 100 milissegundos:

#define joystick1X A0

#define joystick1Y A1

void setup() {

pinMode(joystick1X, INPUT);

pinMode(joystick1Y, INPUT);

Serial.begin(9600);

}

void loop() {

int x, y;

x = analogRead(joystick1X);

y = analogRead(joystick1Y);

Serial.print(x);

Serial.print(" , ");

Serial.print(y);

delay(100);

}

4 - Verifique se todas as funcionalidades se aplicam ao seu projeto.

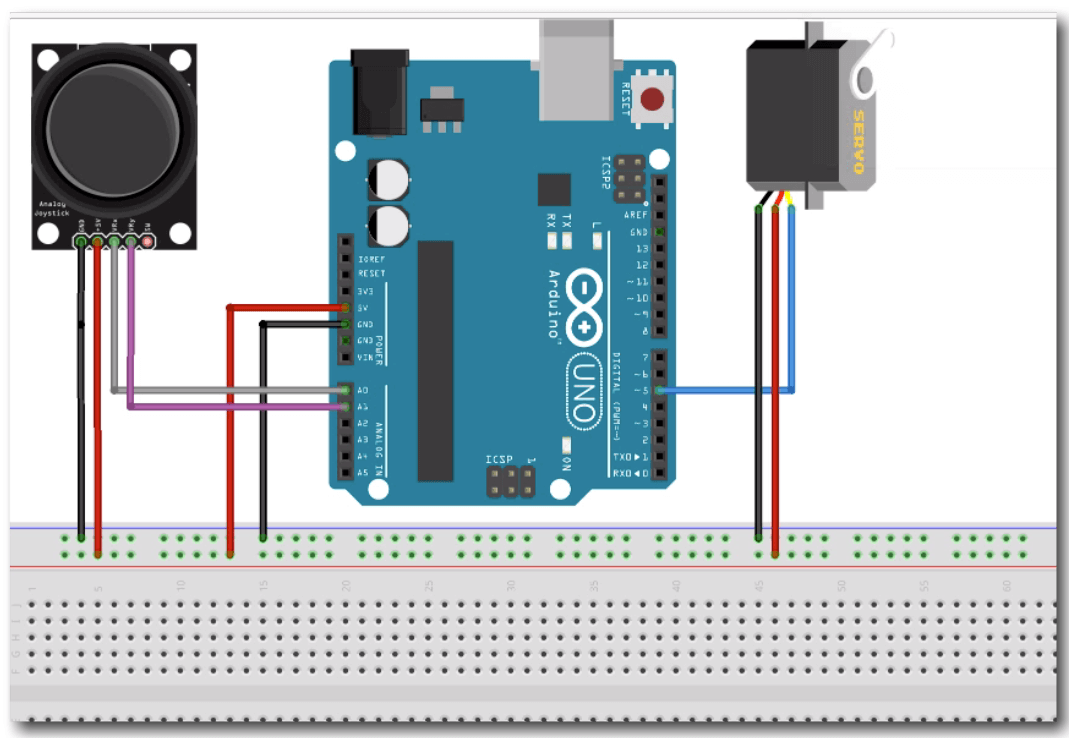
Nessa capítulo, faremos o mapeamento entre a entrada analógica do joystick e a saída do servo motor. Ao final, esperamos que ao mexer o eixo X ou Y do joystick, que a posição do motor seja alterada.

Então vamos entender as conexões.

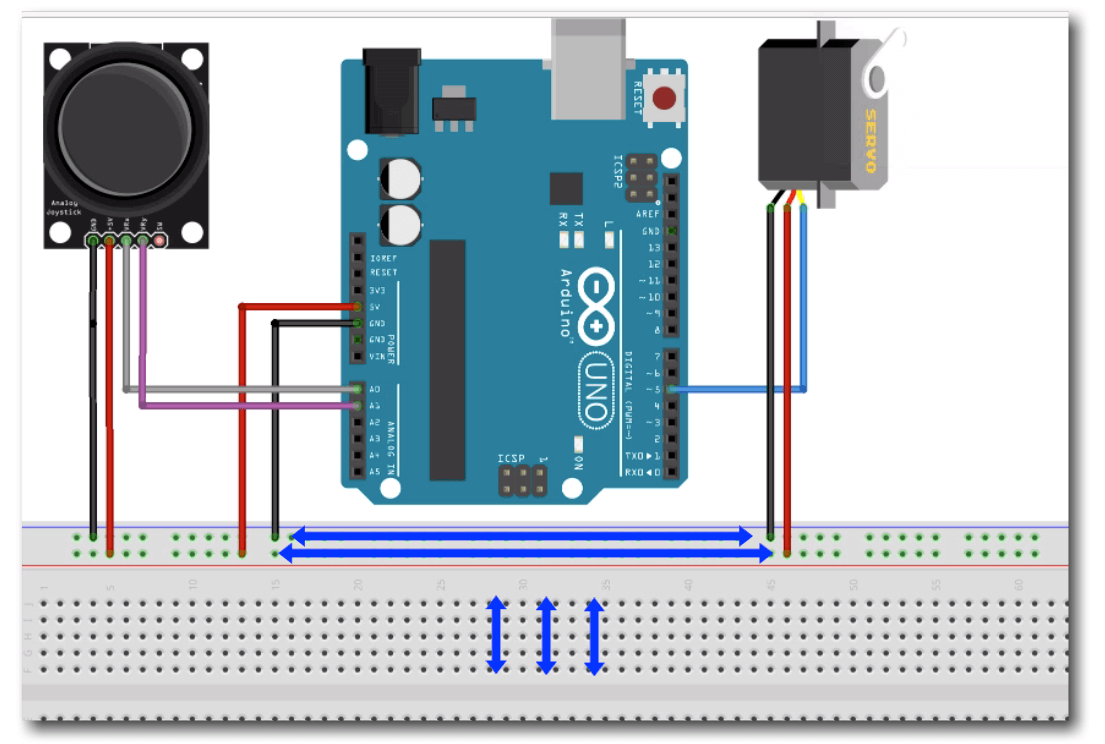
## Desenho do projeto

Ligaremos o **eixo X** do joystick à **entrada analógica A0** do Arduino e o **eixo Y** à **entrada analógica A1**, como vimos no vídeo anterior. Por enquanto, vamos utilizar apenas um joystick e um motor, e usar apenas o eixo X no joystick, apesar de conectarmos os dois eixos (X e Y) ao Arduino. Quando ligarmos o segundo joystick, utilizaremos as entradas analógias **A2** e **A3** para os seus eixos **X** e **Y**, respectivamente.

Já o motor, ligaremos ao **pino 5**, que é um pino PWM. O projeto ficará assim:



Para o nosso projeto, utilizaremos 6 módulos (4 servo motores e 2 joysticks), logo não temos alimentação no Arduino o suficiente para todos eles, isto é, não há pinos de 5V o suficiente. Por isso utilizaremos a protoboard, ligaremos o **5V** do Arduino a uma linha da protoboard e o **GND** a outra. As ligações do meio da protoboard são contínuas na **vertical**, e as linhas de cima, são contínuas na **horizontal**:

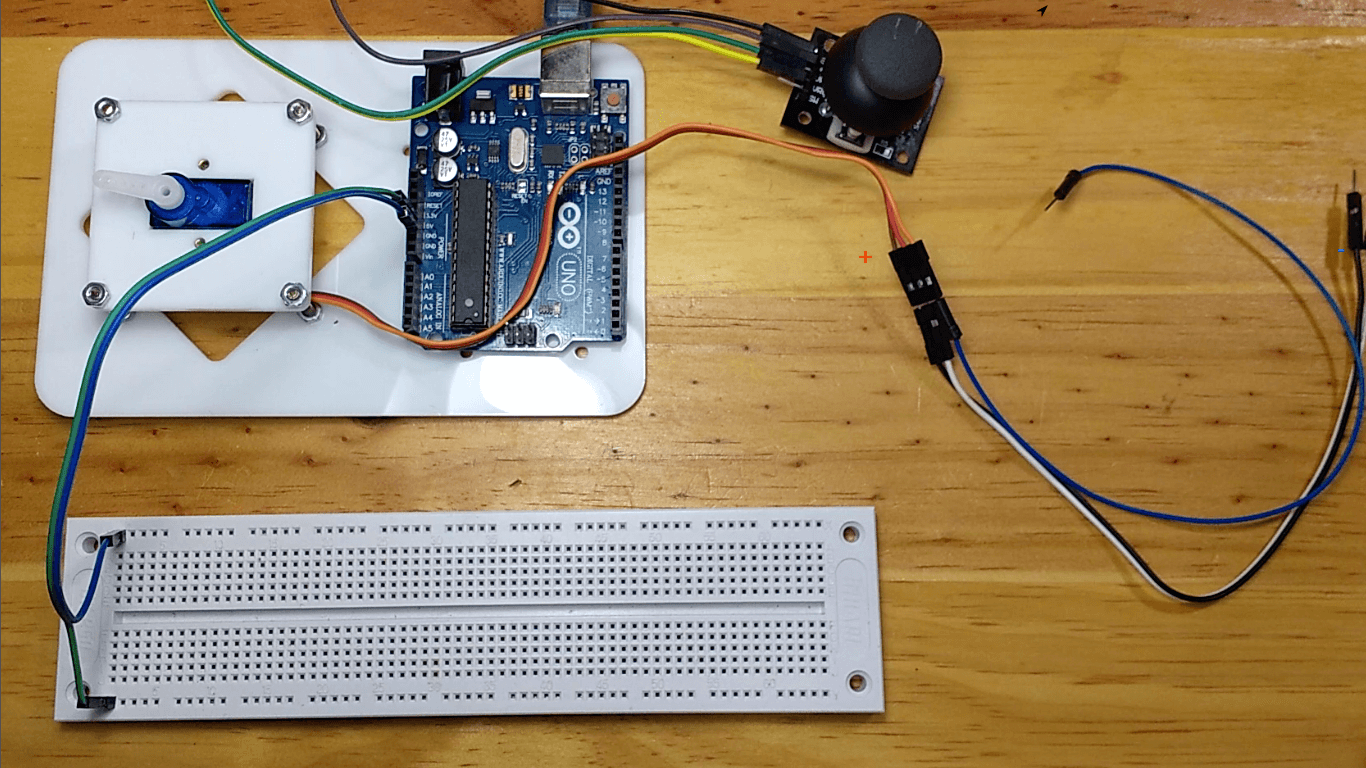


Ou seja, um componente que está ligado a uma ponta da linha estará em contato com outro componente ligado à outra ponta. Logo, alimentando a primeira linha com o GND, podemos conectar os GNDs dos nossos módulos a essa linha, e a mesma coisa vale para os 5V. Assim, basta conectar os 5V e os GNDs dos nossos módulos às suas respectivas linhas da protoboard.

Com escopo do projeto pronto, podemos montá-lo. A primeira coisa que podemos fazer é montar a **base do braço-robô**, somente ela, conectando um motor à mesma.

Em relação ao servo motor, utilizaremos três *jumpers* Macho x Macho para deixar os fios mais extensos. Para o joystick, utilizaremos quatro *jumpers* Macho x Fêmea, e precisaremos de mais dois *jumpers* Macho x Macho, para realizar a conexão do Arduino com a protoboard.

Primeiramente, ligamos um *jumper* Macho x Macho no **pino 5V** do Arduino, e um *jumper* Macho x Macho no **pino GND**. Conectamos a outra extremidade do *jumper*relativo ao GND na primeira linha da protoboard, e a outra extremidade do *jumper*relativo à alimentação na segunda linha da protoboard:



**Conectando o servo motor**

Cada fio possui uma representação, vamos lembrar:

* **Laranja**: fio de controle do motor, que é conectado a um pino PWM do Arduino
* **Vermelho**: fio de alimentação
* **Marrom**: fio terra (GND)

Logo, fazemos as seguintes conexões:

* **Laranja**: é ligado ao pino PWM 5 do Arduino
* **Vermelho**: é ligado à protoboard, à linha alimentada pelos 5V do Arduinho
* **Marrom**: é ligado à protoboard, à linha alimentada pelo GND do Arduinho

**Conectando o joystick**

Os pinos do joystick consistem em:

* **GND**: Terra
* **+5V**: 5 volts, alimentação
* **VRx**: Eixo X do joystick
* **VRy**: Eixo Y do joystick
* **SW**: Saída Digital Eixo Z, quando o botão é pressionado

Lembrando que não utilizaremos o pino **SW**, então fazemos as seguintes conexões:

* **GND**: é ligado à protoboard, à linha alimentada pelo GND do Arduinho
* **+5V**: é ligado à protoboard, à linha alimentada pelos 5V do Arduinho
* **VRx**: é ligado à entrada analógica A0 do Arduino
* **VRy**: é ligado à entrada analógica A1 do Arduino

Feita a montagem, vamos partir para a programação, que será feita no próximo vídeo :)

Na IDE do Arduino, podemos implementar o programa, visando o mapeamento entre o servo motor e o joystick.

A primeira coisa que podemos fazer é importar a biblioteca do servo motor:

#include <Servo.h>

void setup() {

}

void loop() {

}

Vamos definir um ângulo inicial para o servo motor, que será de 90º. Além disso, vamos criar constantes para mapear os pinos dos eixos X e Y do joystick:

#include <Servo.h>

#define ANGULO\_INICIAL\_MOTOR 90

// --- Mapeamento dos Joysticks ---

#define joystick1X A0

#define joystick1Y A1

void setup() {

}

void loop() {

}

Resta agora definir o servo motor. Definiremos o primeiro como **motorBase**:

#include <Servo.h>

#define ANGULO\_INICIAL\_MOTOR 90

// --- Mapeamento dos Joysticks ---

#define joystick1X A0

#define joystick1Y A1

// --- Mapeamento dos Servos ---

Servo motorBase;

void setup() {

}

void loop() {

}

Dentro da função **setup()**, precisamos definir o pino está o controle do servo motor, o pino **5**:

void setup() {

motorBase.attach(5);

}

Ainda nesse função, vamos definir as portas do joystick como portas de entrada. Por enquanto só faremos o mapeamento do pino X:

void setup() {

motorBase.attach(5);

pinMode(joystick1X, INPUT);

}

E colocamos o motor na posição de 90º, guardada na constante **ANGULO\_INICIAL\_MOTOR**:

void setup() {

motorBase.attach(5);

pinMode(joystick1X, INPUT);

motorBase.write(ANGULO\_INICIAL\_MOTOR);

}

Até aqui nenhuma novidade, vamos então iniciar o mapeamento da entrada analógica para o servo motor.

## Mapeamento da entrada analógica para o servo motor

Dentro da funçãon **loop()**, vamos criar a variável **posX**, que irá receber o valor do eixo X do joystick:

void loop() {

int posX = analogRead(joystick1X);

}

Agora precisamos mapear o intervalo numérico da entrada analógica (0 até 1023) para o intervalo de ângulos do servo motor (0 até 180). Para isso existe a função **map()**, que permite efetuar o mapeamento de um intervalo numérico em outro intervalo numérico desejado. Isso significa que em um intervalo numérico, que vai de um valor mínimo até um valor máximo, o valor mínimo será mapeado em um novo valor mínimo, e o valor máximo será mapeado em um novo valor máximo, assim como os valores intermediários, que serão remapeados em novos valores intermediários, de forma correspondente.

No nosso caso, vamos mapear o intervalo 0-1023 para o intervalo 0-180. A função recebe por parâmetro o número a ser mapeado (no nosso caso, o número contido na variável **posX**), o limite inferior do intervalo de valores atual (**0**), o limite superior do intervalo de valores atual (como o valor de **posX** vem da entrada analógica, o limite superior é **1023**), além do limite inferior do novo intervalo de valores mapeados (**0**) e do limite superior do novo intervalo de valores mapeados (**180**, maior ângulo possível do servo motor).

Então, o código fica assim:

void loop() {

int posX = analogRead(joystick1X);

posX = map(posX, 0, 1023, 0, 180);

}

Agora que **posX** representa um ângulo válido, vamos passá-lo para o servo motor e adicionar um delay para garantir um intervalo entre as leituras:

void loop() {

int posX = analogRead(joystick1X);

posX = map(posX, 0, 1023, 0, 180);

motorBase.write(posX);

delay(100);

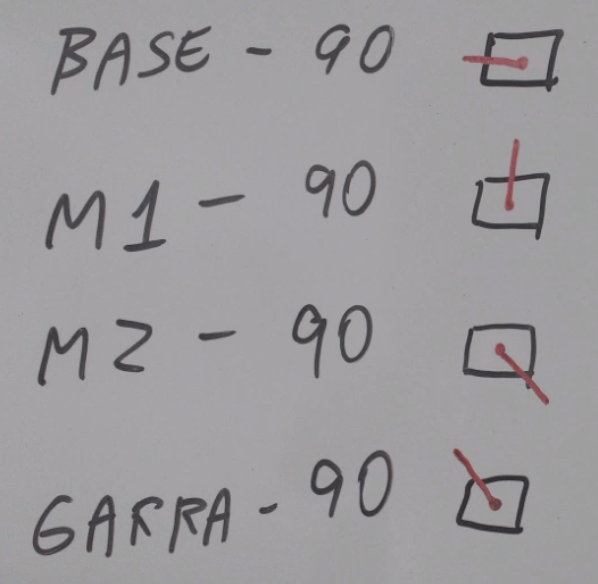
}

Chegou a hora de montarmos o nosso braço-robô.

Podemos seguir o manual, mas há algumas dicas a se seguir. Como foi falado no início do treinamento, recomendamos o braço-robô de **acrílico**, pois o de MDF desgasta facilmente, só é preciso ter atenção na hora de apertar os parafusos, pois o acrílico pode quebrar. Além disso, também é importante remover o plástico de proteção das peças, principalmente as que têm atrito.

No código, daremos nomes aos motores: o da base se chamará **motorBase**, o da garra será chamado de **motorGarra**. Os motores laterais, o que está preso à peça no formato de um retângulo reto, é o **motorBraco2**, e o outro é o **motorBraco1**.

É importante, antes de mais nada, posicionar todos os motores em 90º. No caso das pás dos motores, elas foram posicionadas nas seguintes angulações:



E o intervalo de ângulos em que os motores funcionarão serão os seguintes:

* **motorBase**: de 0º a 180º
* **motorBraco1**: de 45º a 135º
* **motorBraco2**: de 50º a 90º
* **motorGarra**: de 90º a 130º

Lembrando que tudo isso pode variar de acordo com o modelo do braço-robô.

Agora chegou a hora de montar todo o braço junto com os motores. Os passos de montagem variam entre os modelos disponíveis no mercado mas aconselhamos sempre seguir o manual.

Verifique se o seu modelo possui um manual, caso não tenha, segue o link do manual do braço robô que usamos no curso, mas verifique antes se você adquiriu o mesmo modelo: [LINK DO MANUAL](https://s3.amazonaws.com/caelum-online-public/arduino-robotica/bracoslim.pdf)

Para implementar o programa final do braço-robô, vamos utilizar como base o código já feito no aula 4:

#include <Servo.h>

#define ANGULO\_INICIAL\_MOTOR 90

// --- Mapeamento dos Joysticks ---

#define joystick1X A0

#define joystick1Y A1

// --- Mapeamento dos Servos ---

Servo motorBase;

void setup() {

motorBase.attach(5);

pinMode(joystick1X, INPUT);

motorBase.write(ANGULO\_INICIAL\_MOTOR);

}

void loop() {

int posX = analogRead(joystick1X);

posX = map(posX, 0, 1023, 0, 180);

motorBase.write(posX);

delay(100);

}

Vamos começar definindo os pinos do segundo joystick, que serão os pinos **A2** e **A3**, para os eixos X e Y, respectivamente:

// --- Mapeamento dos Joysticks ---

#define joystick1X A0

#define joystick1Y A1

#define joystick2X A2

#define joystick2Y A3

No mapeamento dos servos, vamos adicionar o restante dos motores:

// --- Mapeamento dos Servos ---

Servo motorBase;

Servo motorBraco1;

Servo motorBraco2;

Servo motorGarra;

Agora, vamos criar uma função que mapeia os pinos dos servo motores, eles ficarão nos seguintes pinos:

* **motorBase**: pino **5**
* **motorBraco1**: pino **6**
* **motorBraco2**: pino **9**
* **motorGarra**: pino **10**

Logo, a função ficará assim:

void mapearPinosDosMotores() {

motorBase.attach(5);

motorBraco1.attach(6);

motorBraco2.attach(9);

motorGarra.attach(10);

}

Para finalizar este vídeo, vamos criar uma função para definir o ângulo inicial de todos os motores para 90º, lembrando que já temos a constante **ANGULO\_INICIAL\_MOTOR** com esse valor guardado:

void inicializarAngulosDosMotores() {

motorBase.write(ANGULO\_INICIAL\_MOTOR);

motorBraco1.write(ANGULO\_INICIAL\_MOTOR);

motorBraco2.write(ANGULO\_INICIAL\_MOTOR);

motorGarra.write(ANGULO\_INICIAL\_MOTOR);

}

Para finalizar a inicialização do nosso braço-robô, vamos criar uma função que define os joysticks como **INPUT**:

void mapearJoysticksComoInput() {

pinMode(joystick1X, INPUT);

pinMode(joystick1Y, INPUT);

pinMode(joystick2X, INPUT);

pinMode(joystick2Y, INPUT);

}

Agora que criamos as funções de mapeamento e inicialização, podemos implementar as funções de movimento do braço-robô.

## Funções de locomoção

Chegou a hora de criarmos as funções de mapeamento joystick-servo motor, que farão os movimentos do braço-robô.

Vamos começar pela função responsável pode mover o motor da base, que já tem o código pronto, dentro da função **loop()**. Vamos exportar esse código para dentro da função **moverBase()**:

void moverBase() {

int posX = analogRead(joystick1X);

posX = map(posX, 0, 1023, 0, 180);

motorBase.write(posX);

}

A função que move o **motorBraco1** é basicamente a mesma, mas ela trabalhará com o eixo Y do primeiro joystick, e com os ângulos vistos no primeiro vídeos desta aula:

void moverBraco1() {

int posY = analogRead(joystick1Y);

posY = map(posY, 0, 1023, 45, 135);

motorBraco1.write(posY);

}

Assim, podemos implementar as funções restantes, **moverBraco2()** (que trabalha com o eixo X do segundo joystick) e **moverGarra()** (que trabalha com o eixo Y do segundo joystick):

void moverBraco2() {

int posX = analogRead(joystick2X);

posX = map(posX, 0, 1023, 50, 90);

motorBraco2.write(posX);

}

void moverGarra() {

int posY = analogRead(joystick2Y);

posY = map(posY, 0, 1023, 90, 130);

motorGarra.write(posY);

}

Para finalizar o nosso código, primeiramente devemos chamar as funções de inicialização e mapeamento dentro da função **setup()**. Ela ficará assim:

void setup() {

mapearPinosDosMotores();

inicializarAngulosDosMotores();

mapearJoysticksComoInput();

}

Por fim, chamamos as funções de movimentos dos servo motores dentro da função **loop()**, e adicionamos um *delay* para garantir a leitura dos valores:

void loop() {

moverBase();

moverBraco1();

moverBraco2();

moverGarra();

delay(100);

}

Feito isso, o nosso código está pronto. Na próxima aula, faremos todas as ligações dos joysticks e dos servo motores, e testaremos o nosso código.

Repare que as funções setup e loop chamam apenas sub-funções de mais baixo nível. Todos os detalhes de leitura e escrita estão nas sub-funções, e setup e loop delegam o trabalho. Assim o código fica mais legível e com uma organização melhor.

Segue também o código completo, para sua comparação:

#include <Servo.h>

#define ANGULO\_INICIAL\_MOTOR 90

// --- Mapeamento dos Joysticks ---

#define joystick1X A0

#define joystick1Y A1

#define joystick2X A2

#define joystick2Y A3

// --- Mapeamento dos Servos ---

Servo motorBase;

Servo motorBraco1;

Servo motorBraco2;

Servo motorGarra;

void setup() {

mapearPinosDosMotores();

inicializarAngulosDosMotores();

mapearJoysticksComoInput();

}

void loop() {

moverBase();

moverBraco1();

moverBraco2();

moverGarra();

delay(100);

}

void mapearPinosDosMotores() {

motorBase.attach(5);

motorBraco1.attach(6);

motorBraco2.attach(9);

motorGarra.attach(10);

}

void inicializarAngulosDosMotores() {

motorBase.write(ANGULO\_INICIAL\_MOTOR);

motorBraco1.write(ANGULO\_INICIAL\_MOTOR);

motorBraco2.write(ANGULO\_INICIAL\_MOTOR);

motorGarra.write(ANGULO\_INICIAL\_MOTOR);

}

void mapearJoysticksComoInput() {

pinMode(joystick1X, INPUT);

pinMode(joystick1Y, INPUT);

pinMode(joystick2X, INPUT);

pinMode(joystick2Y, INPUT);

}

void moverBase() {

int posX = analogRead(joystick1X);

posX = map(posX, 0, 1023, 0, 180);

motorBase.write(posX);

}

void moverBraco1() {

int posY = analogRead(joystick1Y);

posY = map(posY, 0, 1023, 45, 135);

motorBraco1.write(posY);

}

void moverBraco2() {

int posX = analogRead(joystick2X);

posX = map(posX, 0, 1023, 50, 90);

motorBraco2.write(posX);

}

void moverGarra() {

int posY = analogRead(joystick2Y);

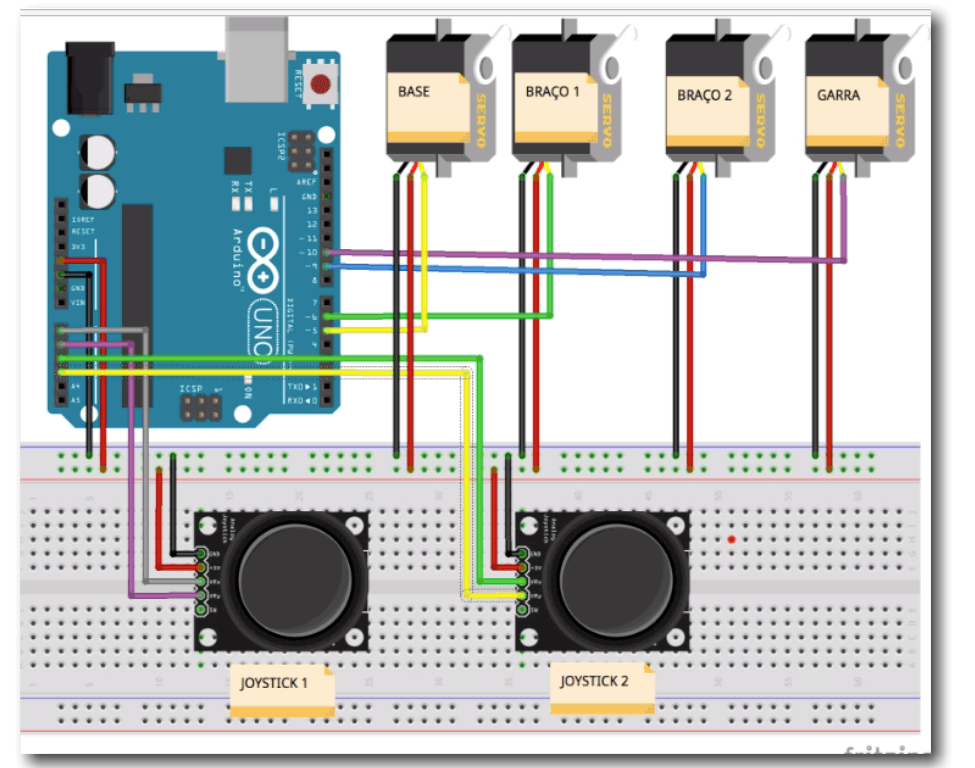
posY = map(posY, 0, 1023, 90, 130);

motorGarra.write(posY);

}

Chegou a hora de realizar todas as conexões do nosso braço-robô.

Nós já levamos a alimentação (5V) e o terra (GND) do Arduino até a protoboard, e dela nós fazemos as ligações que os servo motores e joysticks necessitam. A figura abaixo ilustra um pouco isso:



Então, pegamos quatro *jumpers*, conectamos um em cada **fio vermelho** dos quatro servo motores, e os conectamos na linha **positiva** da protoboard, ou seja, a linha alimentada pelo **5V** do Arduino. Para o terra, fazemos a mesma coisa, pegamos quatro *jumpers*, conectamos um em cada **fio marrom** dos quatro servo motores, e os conectamos na linha **negativa** da protoboard, ou seja, a linha alimentada pelo **GND** do Arduino.

Em relação aos fios laranjas dos servo motores, que são os fios de controle, eles serão conectados aos seguintes pinos do Arduino:

* Controle do **motorBase**: pino **5**
* Controle do **motorBraco1**: pino **6**
* Controle do **motorBraco2**: pino **9**
* Controle do **motorGarra**: pino **10**

No próximo vídeo, terminaremos todas as conexões do nosso braço-robô :)

Antes de partir para o teste final do projeto, você pode esquematizá-lo através do Fritzing. Isso é totalmente opcional, mas é uma abordagem recomendada para que você se sinta ainda mais seguro quando for montar o circuito.

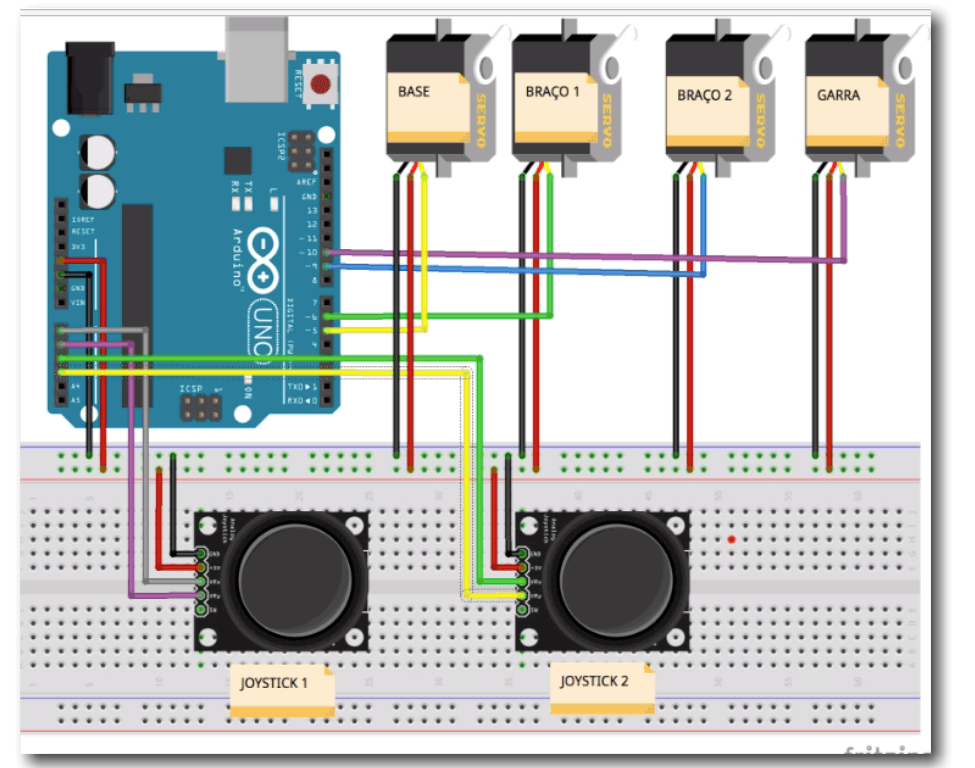
Lembrando, os motores usam os pinos digitais:

* Controle do motorBase: pino 5
* Controle do motorBraco1: pino 6
* Controle do motorBraco2: pino 9
* Controle do motorGarra: pino 10

Os joysticks usam os pinos analógicos:

* Joystick 1, eixo X: A0
* Joystick 1, eixo Y: A1
* Joystick 2, eixo X: A2
* Joystick 2, eixo Y: A3

Além disso, cada componente precisa da alimentação (5V) e o fio terra (GND).



Completamos nosso projeto, mas existem coisas que ainda podem ser melhoradas.

Atualmente, nosso braço robo tem um comportamento um pouco instável.

Quando mexemos no analógico, nosso servo motor se movimenta muito rápido, o que pode fazer com que nosso braço caia.

Além disso, será que realmente queremos que nosso robo toda vez retorne para a posição inicial? Não seria mais interessante manter a posição de acordo com que mexemos no analógico?

Primeiramente, devemos entender o motivo de tudo isso.

Quando alteramos a posição do analógico, pedimos especificamente para que o arduino posicione o servo na leitura analógica que está sendo passado, e isso acontece de maneira muito rápida.

É importante ressaltar que com o analógico na posição inicial, também estamos passando informação ao arduino e é isso que faz com que nosso braço sempre retorne.

Se o analógico retorna, o braço também retorna!

Então chegou a hora de resolvermos esses problemas e melhorar nosso projeto.

Para isso, faremos uso de uma biblioteca bastante útil, chamada VarSpeedServo, com ela podemos modificar a velocidade de um servo motor de maneira muito simples.

1) Faça o download abaixo:

[Clique Aqui!](https://github.com/netlabtoolkit/VarSpeedServo/releases/download/v1.1.3/VarSpeedServo.zip)

2) Após feito o download, vá em **SKETCH**, após isso, selecione **IMPORT LIBRARY** e clique em **ADD LIBRARY**.

3) Procure pelo arquivo baixado **.zip**

Isso instalará a biblioteca nos seus "Meus Documentos" ou "Documentos", em arduino/libraries

Utilizando essa biblioteca, **algumas coisas serão diferentes**.

Quando fizermos meuServo.write(), poderemos passar um **segundo valor**, que será a **velocidade**. Veremos mais a fundo na hora da mudança do código.

Agora com nossa biblioteca devidamente instalada, vamos começar a alterações na próxima atividade.

## Vamos modificar nosso código focando no motorBase.

Antes de tudo, vamos importar a biblioteca, para isso, escreva:

#include <VarSpeedServo.h>

**A linha acima deve vir antes de qualquer coisa!**

Após importar a biblioteca VarSpeedServo, vamos criar uma variável auxiliar para cada servo motor, logo abaixo do nosso **define** ANGULO\_INICIAL\_MOTOR

int auxBase = ANGULO\_INICIAL\_MOTOR;

Essa será a variável que vai manter nosso braço parado!

A segunda moficação acontece no nosso mapeamento dos servos, agora teremos o seguinte:

.

.

VarSpeedServo motorBase;

.

.

Agora modificaremos o nosso método, mas antes vamos dar uma revisada.

Atualmente, temos o seguinte:

Em linguagem humana, lemos nosso analógico, mapeamos essa leitura, transformando-a em um valor dentro do intervalo entre 0 e 180, e então posicionamos nosso motor nessa posição:

void moverBase(){

int posX = analogRead(joystick1X);

posX = map(posX, 0, 1023, 0, 180); //aqui a mágica acontece

motorBase.write(posX);

}

Como vimos, isso acontece de maneira muito rápida e não mantém nosso braço parado, ele sempre retorna para o angulo inicial.

Analisando bem a situação, podemos ver que isso ocorre devido ao loop do arduino, que está o tempo todo chamando todos os métodos e lendo a posição dos analógicos

Para resolver esse problema, teremos que seguir a seguinte linha de raciocínio:

Teremos a leitura inicial do joystick, mas não posicionaremos nosso servo diretamente, antes faremos um teste.

Se a leitura do joystick for maior que determinado valor, vamos incrementar a posição do braço, caso contrário, iremos decrementar, vamos começar:

Manteremos as primeiras duas linhas, lemos a posição e mapeamos.

void moverBase(){

int posX = analogRead(joystick1X);

posX = map(posX, 0, 1023, 0, 180);

}

Escreveremos um if onde testaremos a posição do analógico, ficando da seguinte forma:

void moverBase(){

int posX = analogRead(joystick1X);

posX = map(posX, 0, 1023, 0, 180);

if(posX > 100){

// Faz algo

}

}

Devemos também ter uma condicional para o caso de o valor ser menor:

void moverBase(){

int posX = analogRead(joystick1X);

posX = map(posX, 0, 1023, 0, 180);

if(posX > 100){

//Faz algo

}else if(posX < 80){

//Faz algo

}

}

Repare que estamos comparando com 100 e 80. Tendo em vista que nosso analógico inicia no 90, demos uma tolerância, já que o mesmo pode ter variações de acordo com o uso.

Agora devemos completar nosso código, removendo o comentário //Faz algo.

Dentro dos **ifs**, temos que incrementar ou decrementar nossa posição, utilizando a nossa variável auxiliar, nesse caso **auxBase**.

void moverBase(){

int posX = analogRead(joystick1X);

posX = map(posX, 0, 1023, 0, 180);

if(posX > 100){

auxBase+=10;

motorBase.write(auxBase,50);

}else if(posX < 80){

auxBase-=10;

motorBase.write(auxBase,50);

}

É de extrema importância que você teste com diferentes valores e veja o que melhor se encaixa para você. Troque também os valores do segundo parâmetro do write, e veja que a velocidade do braço também muda.

Lembre-se de fazer a alteração em todo seu código e utilizar uma variável correspondente para cada servo!