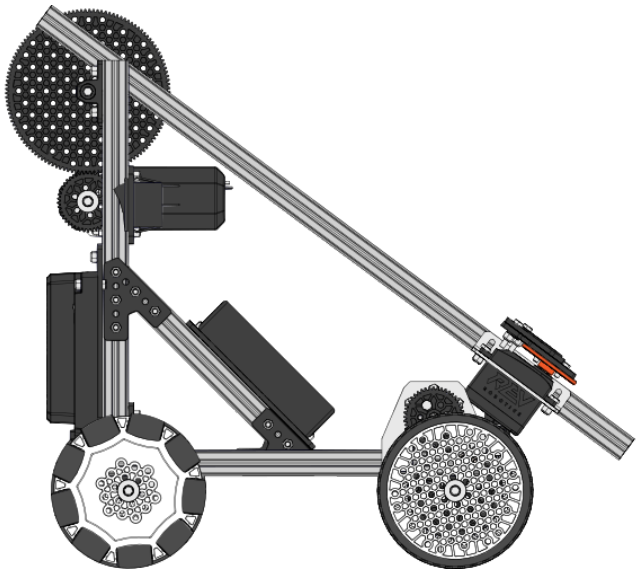
Controle de braço - OnBot Java

Introdução ao controle de braço

O controle de robôs assume muitas formas diferentes. Agora que você passou pela programação de um drivetrain, podemos aplicar esses conceitos ao controle de outros mecanismos. Como este guia utiliza a classe Bot, o foco será nos conceitos básicos de controlar o seu mecanismo principal, um braço de articulação única.



Controlar um braço requer um processo mental diferente daquele que você usou para controlar o drivetrain. Enquanto o drivetrain usa o movimento de rotação dos motores para percorrer uma distância linear, um braço gira em torno de um ponto central, ou junta. Ao trabalhar com um braço, você precisará ter cautela com as limitações físicas do robô, incluindo capacidade de carga, amplitude de movimento e outras forças que possam ser aplicadas.

Nesta seção, você aprenderá a usar os controles do D-pad do gamepad e o Sensor de Toque instalado para controlar o braço. No entanto, o foco desta seção é usar o código para limitar a amplitude de movimento do braço.

Seção	Objetivos da seção
Noções básicas de programação sobre o braço	Introdução à codificação de um braço para controle teleoperado e trabalho com um interruptor de limite.
Programando um braço para uma posição	Utilizando codificadores de motor para mover um braço para uma posição específica, como de 45 graus para 90 graus.
Utilizando limites para controlar a faixa do motor	Trabalhando com os fundamentos do controle de braço, codificador de motor e interruptores de limite para controlar a amplitude de movimento de um braço.

Noções básicas de programação sobre o braço

Comece criando um modo operacional básico chamado HelloRobot_ArmControl.

> Para obter mais informações sobre como criar um modo operacional (op mode), consulte a seção Test Bed - Onbot Java.

Diferentemente do joystick, que envia valores correspondentes à posição do joystick, o Dpad no gamepad envia valores booleanos FALSE/TRUE. Para determinar como o braço se move quando você pressiona DpadUp ou DpadDown, é necessário usar uma instrução if/else if. Crie uma instrução if/else if semelhante à abaixo:

```
while (opModeIsActive()) {
   if(gamepad1.dpad_up){


               }
   else if (gamepad1.dpad_down){


               }
     }
```

Agora que a estrutura básica está no lugar, podemos adicionar os blocos necessários para ditar a direção do braço. A melhor prática é fazer com que o braço se mova para cima quando DpadUp é selecionado e para baixo quando DpadDown é selecionado. Para fazer isso, vamos adicionar arm.setPower(); a cada parte executável da instrução if/else if.

> Lembre-se de que o valor atribuído a setPower dita a direção e a velocidade do motor. Entre o motor e a engrenagem no robô classe, um valor positivo moverá o braço para cima, enquanto um valor negativo moverá o braço para baixo. Se você não tem certeza sobre a direção do seu motor, crie o seguinte código e teste para garantir que o motor esteja se comportando conforme o esperado:

```
if(gamepad1.dpad_up){
       arm.setPower(0.2);
            }
else if (gamepad1.dpad_down){
       arm.setPower(-0.2);
            }
```

> Iniciar com um ciclo de trabalho mais baixo, como o 0,2 mostrado no código acima, permitirá testes mais fáceis ao tomar decisões para o braço. Mais tarde, neste guia, alteraremos para um ciclo de trabalho mais alto.

> Guarde o modo de operação e tente executar o código. Considere as seguintes perguntas.
>
> - O que acontece se você pressionar para cima no Dpad?
> - O que acontece se você pressionar para baixo no Dpad?

Atualmente, a lógica da instrução if/else if declara que quando gamepad1.dpad_up é verdadeiro (foi pressionado), o motor irá funcionar na direção para frente (ou, neste caso, para cima) a 20% de ciclo de trabalho. Se gamepad1.dpad_down for verdadeiro, o motor irá funcionar reversamente a 20% de ciclo de trabalho. Se você executou o código até este ponto, pode ter notado que mesmo quando você soltou o Dpad, o motor continuou a funcionar na direção selecionada. A instrução if/else if atual diz ao robô quando o motor deve se mover e em que direção, mas nada diz ao motor para parar, assim o braço continua a se mover sem limites.

Para corrigir isso, edite a instrução if/else if para incluir uma ação a ser realizada se nenhuma das condições do gamepad for verdadeira. Como queremos que o braço pare de se mover se nenhuma das condições do gamepad for atendida, vamos usar arm.setPower(0); para parar o motor.

```
if(gamepad1.dpad_up){
       arm.setPower(0.2);
            }
else if (gamepad1.dpad_down){
       arm.setPower(-0.2);
```

```
            }
else {
       arm.setPower(0);
            }
```

> Tente salvar e executar novamente o modo de operação. Preste atenção na velocidade do braço subindo em comparação com descendo. A velocidade parece a mesma?

Trabalhar com um braço introduz diferentes fatores a serem considerados em comparação com o que você viu anteriormente com trens de acionamento. Por exemplo, você notou alguma diferença nas velocidades ao mover o braço para cima ou para baixo? Ao contrário do trem de acionamento, onde o efeito da gravidade impacta os motores de maneira consistente em qualquer direção, a gravidade desempenha um papel significativo na velocidade do motor do braço.

Adicionando um interruptor de limite

Outra consideração a ser feita são as limitações físicas do mecanismo do braço. Certos mecanismos podem ter uma limitação física, e quando essa limitação é ultrapassada, há o risco de danificar o mecanismo ou outro componente do robô. Existem algumas maneiras de limitar o mecanismo com sensores que ajudarão a reduzir a possibilidade de um mecanismo ultrapassar suas limitações físicas. Nesta seção, vamos nos concentrar no uso de um interruptor de limite para restringir a faixa de movimento do braço.

> Esta seção pressupõe que você tenha um conhecimento básico sobre interruptores de limite a partir da seção de Testes e da artigo sobre Sensores Digitais.

Como você pode se lembrar da seção Testes, os interruptores de limite usam lógica booleana para indicar quando um limite foi atingido. Os interruptores de limite geralmente se apresentam na forma de sensores digitais, como o Sensor de Toque, já que os sensores digitais reportam um sinal booleano de ligado/desligado para o sistema, assim como um interruptor de luz.

Se você estiver usando um robô da Classe Bot, seu robô deve ter um Sensor de Toque montado na frente do chassi. Você também deve ter instalado um Limit Switch Bumper. Juntos, esses itens formam um sistema de interruptor de limite. Ao utilizar o sistema de interruptor de limite, você pode evitar que o braço do seu robô Classe Bot ultrapasse o limite físico inferior, ou o que será conhecido como nossa posição inicial. Vamos começar a programar!

> Antes de prosseguir com o código, certifique-se de que o mecanismo está interagindo com e pressionando o Sensor de Toque. Se você estiver usando a Classe Bot, isso envolve garantir que o chassi esteja pressionando ativamente o Sensor de Toque quando o braço descer.

No trecho "Banco de testes - Onbot Java", você aprendeu como criar um programa básico de interruptor de limite, semelhante ao exemplo abaixo.

```
 if (touch.getState()){
    //Touch Sensor is not pressed
     arm.setPower(0.2);


} else {
    //Touch Sensor is pressed
    arm.setPower(0);
                        }
```

Se você se recorda da seção inicial sobre interruptores de limite, o Sensor de Toque opera em um estado binário de FALSO/VERDADEIRO. Quando o sensor de toque não está pressionado, touch.getState() retorna verdadeiro; quando o sensor de toque está pressionado, touch.getState() retorna falso. A lógica do código afirma que quando o sensor de toque não está pressionado, o motor funciona com um ciclo de trabalho de 20%.

Em vez de fazer o motor funcionar com um ciclo de trabalho de 20% quando o Sensor de Toque não está pressionado e parar quando o sensor é pressionado, queremos controlar o braço usando o gamepad ainda. Para fazer isso, podemos aninhar a instrução if/else if do Gamepad dentro da instrução if/else do Interruptor de Limite.

> Para esta próxima parte, estaremos utilizando a instrução if/else if criada nas seções Basics of Programming and Arm. Daqui para frente, essa lógica básica de código será referida como a instrução if/else if do Gamepad. O código do interruptor de limite será conhecido como a instrução if/else do Interruptor de Limite. Ambas as partes do código serão referenciadas novamente.

```
if(gamepad1.dpad_up){
       arm.setPower(0.2);
            }
else if (gamepad1.dpad_down){
       arm.setPower(-0.2);
            }
else {
       arm.setPower(0);
            }
```

Segundo código:

```
if(touch.getState()){
      if(gamepad1.dpad_up){
            arm.setPower(0.2);
                  }
      else if (gamepad1.dpad_down){
            arm.setPower(-0.2);
                  }
      else {
            arm.setPower(0);
                  }
       }
  else {
      arm.setPower(0);
       }
```

> Salve o modo operacional e execute-o. O que acontece quando o Sensor de Toque é pressionado?

Uma das características comuns de um interruptor de limite, como o sensor de toque, é a capacidade de redefinir para o seu estado padrão. Se você pressionar o sensor de toque com o dedo, pode notar que assim que liberar a pressão que está aplicando, o sensor de toque voltará ao seu estado padrão de "não pressionado". No entanto, é necessário liberar a pressão para conseguir isso.

> Certifique-se de que o mecanismo está efetivamente interagindo com o Sensor de Toque. Para o Bot da Classe, talvez seja necessário ajustar o Sensor de Toque para que o Limit Switch Bumper esteja interagindo com ele de forma mais consistente.

O código no bloco de informações acima determina que quando o Sensor de Toque é pressionado, o motor do braço é definido como zero. Isso funcionaria em um mecanismo onde o Sensor de Toque pode retornar ao seu estado padrão por conta própria. No entanto, uma vez que o braço pressiona o Sensor de Toque, o peso do mecanismo impedirá que o Sensor de Toque retorne ao seu estado padrão. A combinação do peso do mecanismo e a lógica do código no bloco de informações significa que, uma vez que o braço atinge seu limite, ele não será capaz de se mover novamente.

Para remediar isso, uma ação para mover o braço na direção oposta ao limite precisa ser adicionada à instrução "else". Como o Sensor de Toque é um limite inferior para o braço, o braço precisará se mover para cima (ou o motor na direção para a frente) para se afastar do sensor de toque. Para fazer isso, podemos criar uma instrução if/else semelhante à nossa instrução if/else do gamepad. Em vez de ter as operações normais do gamepad, quando o Sensor de Toque e o DpadUp são pressionados, o braço se move para longe do Sensor de Toque. Uma vez que o Sensor

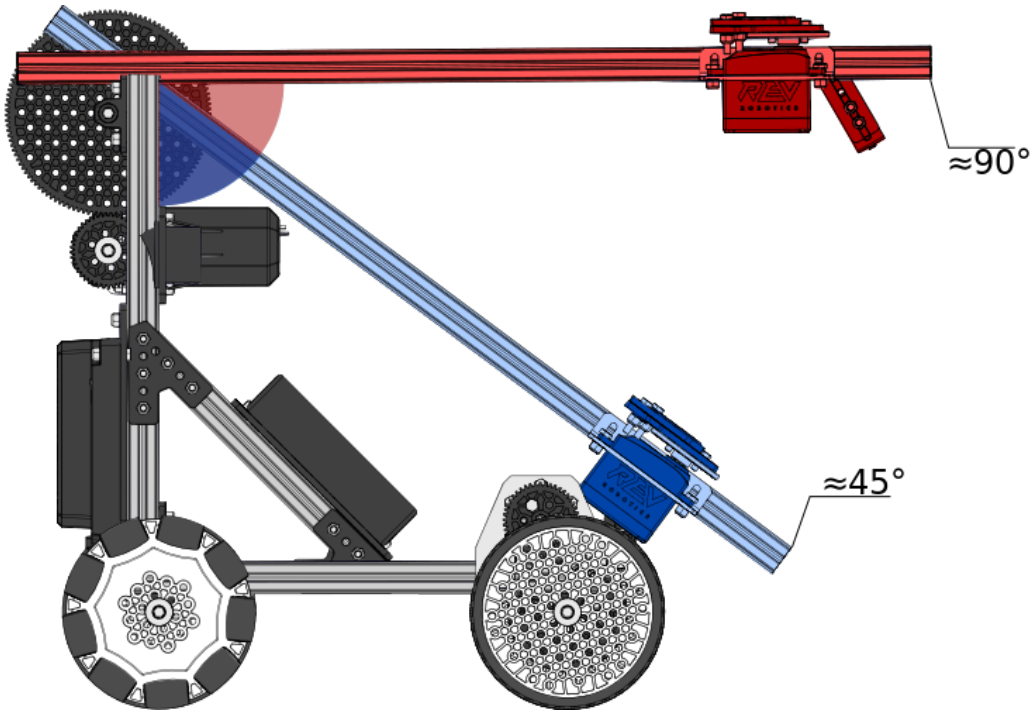
de Toque não relata mais falso, as operações normais do gamepad retornam e o braço pode se mover em qualquer direção novamente.

```
if(touch.getState()){
      if(gamepad1.dpad_up){
            arm.setPower(0.2);
                  }
      else if (gamepad1.dpad_down){
            arm.setPower(-0.2);
                  }
      else {
            arm.setPower(0);
                  }
       }
  else {
       if(gamepad1.dpad_up){
            arm.setPower(0.2);
                  }
       else{
            arm.setPower(0);
       }
```

Programando um braço para uma posição

Na seção de Navegação por Encoder, foi introduzido o conceito de mover o motor para uma posição específica com base em ticks do encoder. O processo destacado na Navegação por Encoder focou em como converter de ticks do encoder para rotações e, consequentemente, para uma distância linear. Um procedimento semelhante pode ser utilizado para mover o braço para uma posição específica. No entanto, ao contrário do drivetrain, o braço não segue um caminho linear. Em vez de converter para uma distância linear, faz mais sentido converter os ticks do encoder em um ângulo medido em graus.

Na imagem abaixo, são apresentadas duas posições potenciais para o braço do ClassBot. Uma das posições - destacada em azul abaixo - é a posição em que o braço encontra o limite do sensor de toque. Devido ao limite, esta posição será nossa posição padrão ou de início. No guia de construção do ClassBot, sabe-se que a extrusão que suporta a bateria está em um ângulo de 45 graus. Como o braço está aproximadamente paralelo a essas extrusões quando está na posição inicial, podemos estimar que o ângulo padrão do braço é aproximadamente 45 graus.



O objetivo desta seção é determinar a quantidade de ticks do encoder necessários para mover o braço de sua posição inicial para uma posição em torno de 90 graus. Existem algumas maneiras diferentes de realizar isso. Uma estimativa pode ser feita movendo o braço para a posição desejada e registrando o retorno de telemetria da Estação do Motorista. Outra opção é realizar os cálculos matemáticos para encontrar a quantidade de ticks do encoder que ocorrem por grau movido. Siga esta seção para percorrer ambas as opções e determinar qual é a melhor para a sua equipe.

Estimando a posição do braço

Para estimar a posição do braço utilizando telemetria e testando, vamos começar com uma programação if/else com controle

```
if(gamepad1.dpad_up){
       arm.setPower(0.2);
            }
else if (gamepad1.dpad_down){
       arm.setPower(-0.2);
            }
else {
       arm.setPower(0);
            }
```

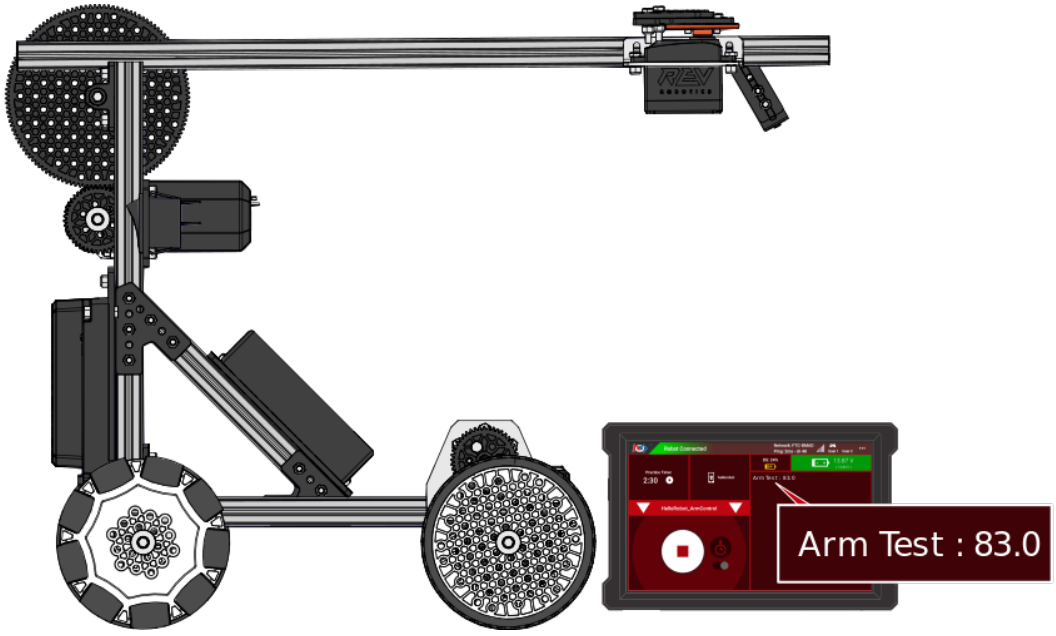
> Por enquanto, você pode comentar o código relacionado ao interruptor de limite.

Dentro do loop while, adicione a linha telemetry.addData("Arm Test", arm.getCurrentPosition()); e telemetry.update();.

```
while(opModeIsActive){
    if(gamepad1.dpad_up){
       arm.setPower(0.2);
            }
     else if (gamepad1.dpad_down){
       arm.setPower(-0.2);
            }
     else {
       arm.setPower(0);
            }
     telemetry.addData("Arm Test", arm.getCurrentPosition());
     telemetry.update();

 }
```

Guarde o modo operacional (op mode) e execute-o. Utilize os comandos do gamepad para mover o braço para a posição de 90 graus. Assim que o braço estiver devidamente posicionado, leia as informações de telemetria na Driver Station para determinar a contagem do codificador em relação à posição do braço.



> Recordem-se da seção Conceitos Básicos de Encoder, na qual a posição do encoder é definida como 0 cada vez que o Control Hub é ligado. Isso significa que, se o braço estiver em uma posição diferente da posição inicial quando o

> Control Hub for ligado, essa posição se tornará zero em vez da posição inicial. O número fornecido na imagem acima não é necessariamente uma contagem precisa do encoder para a posição de 90 graus. Para obter a leitura mais precisa do encoder para o seu robô, certifique-se de que a posição inicial seja registrada como 0 contagens do encoder. Para aumentar ainda mais a precisão, considere realizar várias execuções de teste antes de decidir sobre o número de contagens.

Recordem-se de que, para executar RUN_TO_POSITION, as seguintes três linhas de código precisam ser adicionadas a ambas as seções do bloco if/else if do Gamepad.

```
arm.setTargetPosition(0);
arm.setMode(DcMotor.RunMode.RUN_TO_POSITION);
arm.setPower(0);
```

Quando DpadUp for pressionado, o braço deve se mover para a posição de 90 graus. Quando DpadDown for pressionado, o braço deve voltar à posição inicial. Para fazer isso, defina a primeira linha arm.setTargetPosition(0); igual ao número de ticks necessários para o seu braço atingir 90 graus. Para este exemplo, vamos usar 83 ticks.

Como queremos que DpadDown retorne o braço à posição inicial, manter arm.setTargetPosition(0); definido como 0 nos permitirá realizar isso. Defina ambas as linhas arm.setPower(0); como 0,5.

```
if(gamepad1.dpad_up){
     arm.setTargetPosition(83);
     arm.setMode(DcMotor.RunMode.RUN_TO_POSITION);
     arm.setPower(0.5);
            }
else if (gamepad1.dpad_down){
      arm.setTargetPosition(0);
      arm.setMode(DcMotor.RunMode.RUN_TO_POSITION);
      arm.setPower(0.5);
            }
```

> Observação: o código acima tem uma nome de arquivo "Target Position if/else if", esse código vai ser referênciado de novo

> Recordem-se de que a posição alvo (target position) dita em que direção o motor se move, assumindo o controle da direcionalidade anteriormente controlada por arm.setPower(); Portanto, ambos os blocos podem ser definidos com um valor positivo, já que eles

> controlarão a velocidade.

Se você tentar executar este código, pode perceber que o braço oscila na posição de 90 graus. Quando esse comportamento está presente, você também deve notar a saída de telemetria para as contagens do encoder flutuando. RUN_TO_POSITION é um Controle em Malha Fechada, o que significa que, se o braço não atingir perfeitamente a posição alvo, o motor continuará a oscilar até conseguir. Quando os motores continuam a oscilar e nunca atingem completamente a posição alvo, isso pode ser um sinal de que os fatores que determinam as tolerâncias e outros aspectos do loop fechado não estão ajustados para esse motor ou mecanismo específico. Existem maneiras de ajustar o motor, mas por enquanto queremos nos concentrar em trabalhar com o braço e expandir sobre como limites e posições funcionam em relação ao mecanismo.

Calculando a posição alvo

Na introdução inicial ao "run to position", você trabalhou nos cálculos necessários para converter os ticks por rotação de um motor em ticks por milímetro movido. Agora, queremos nos concentrar em como converter os ticks por rotação do motor em ticks por grau movido. A partir da seção anterior, você deve ter uma estimativa aproximada da quantidade de ticks necessários para chegar à posição de 90 graus. O objetivo desta seção é trabalhar na obtenção de uma posição mais exata.

Para começar, você precisará de algumas das mesmas variáveis que usamos em Navegação com Encoder.

Ticks por revolução

Recordem-se de que os ticks por revolução do eixo do encoder são diferentes dos ticks por revolução do eixo que controla um mecanismo. Vimos isso na seção de Navegação com Encoder, quando os ticks por revolução no motor eram diferentes dos ticks por revolução da roda. À medida que o movimento é transmitido de um motor para um mecanismo, a resolução dos ticks do encoder muda.

> Para mais informação sobre o efeito da transmissão em um mecanismo, consulte a seguinte seção:

Redução

A quantidade de ticks por revolução do eixo do encoder depende do motor e do encoder. Os fabricantes de motores com encoders embutidos terão informações sobre a quantidade de ticks por revolução.

> Visite o site da fabricante dos seus motores e encoders para saber as contagens

Nas especificações do Core Hex tem dois tipos diferentes de Contagens por revolução do encoder:

- No motor - **4 contagens/revolução**
- Na saída - **288 contagens/revolução**

No motor, temos o número de contagens do encoder no eixo em que o encoder está instalado. Esse número é equivalente aos 28 counts por revolução que usamos para o HD Hex Motor. Os 288 counts "na saída" levam em consideração a mudança na resolução após o movimento ser transmitido do motor para a caixa de engrenagens embutida de 72:1. Vamos usar o valor 288 como ticks por revolução para que não seja necessário considerar a caixa de engrenagens em nossa variável total de redução de engrenagens.

Redução total

Como incorporamos a redução de engrenagens da caixa de engrenagens do motor nos ticks por revolução, o foco principal desta seção é calcular a redução de engrenagens da junta do braço. O eixo do motor aciona uma engrenagem de 45 dentes que transmite movimento para uma engrenagem de 125 dentes. A relação total de engrenagens é 125D:45D. Para calcular a redução de engrenagens para este conjunto de engrenagens, podemos simplesmente dividir 125 por 45.

$$125/45 = 2.777778$$

Para recapitular, para o Robô V2 as seguinte informações são verídicas:

Descrição	Valor
Ticks por revolução	288 ticks
Redução total	2.777778

Vamos criar duas variáveis agora que temos essa informação:

- COUNTS_PER_MOTOR_REV
- GEAR_REDUCTION

> A convenção comum de nomeação para variáveis constantes é conhecida como CONSTANT_CASE, em que o nome da variável está todo em maiúsculas e as palavras são separadas por um sublinhado.

Adicione as variáveis COUNTS_PER_MOTOR_REV e GEAR_REDUCTION ao op mode abaixo de onde as variáveis de hardware são criadas.

```
public class HelloRobot_ArmControl extends LinearOpMode {
    private DcMotor arm;
```

```
    static final double     COUNTS_PER_MOTOR_REV    = 288;
    static final double     GEAR_REDUCTION    = 2.7778;
```

Agora que essas duas variáveis foram definidas, podemos usá-las para calcular outras duas variáveis: a quantidade de contagens do encoder por rotação da engrenagem acionada de 125 dentes e o número de contagens por grau movido.

Calcular as contagens por revolução da engrenagem de 125 dentes (ou COUNTS_PER_GEAR_REV) é a mesma fórmula usada na Navegação com Encoder para nossa variável COUNTS_PER_WHEEL_REV. Portanto, para obter essa variável, podemos multiplicar COUNTS_PER_MOTOR_REV por GEAR_REDUCTION.

```
static final double  COUNTS_PER_GEAR_REV    = COUNTS_PER_MOTOR_REV * GEAR_REDUCTION;
```

Para calcular o número de contagens por grau movido (ou COUNTS_PER_DEGREE), divida a variável COUNTS_PER_GEAR_REV por 360.

```
static final double     COUNTS_PER_DEGREE    = COUNTS_PER_GEAR_REV/360;
```

Adicione essas variáveis ao Op Mode.

```
public class HelloRobot_ArmControl extends LinearOpMode {
    private DcMotor arm;

    static final double     COUNTS_PER_MOTOR_REV    = 288;
    static final double     GEAR_REDUCTION    = 2.7778;
    static final double     COUNTS_PER_GEAR_REV    = COUNTS_PER_MOTOR_REV * GEAR_REDUCTION;
    static final double     COUNTS_PER_DEGREE    = COUNTS_PER_GEAR_REV/360;
```

Finalmente, precisamos criar uma variável não constante que atuará como nossa posição. Crie uma variável chamada armPosition acima do comando waitForStart();.

```
public void runOpMode() {
        arm = hardwareMap.get(DcMotor.class, "arm");

        int armPosition;

        waitForStart();
```

Adicione essa variável à seção if(gamepad1.dpad_up) da instrução if/else if, já que essa seção indica a posição de 90 graus. Para chegar à posição de 90 graus, o braço precisa se mover aproximadamente 45 graus. Defina a posição do braço como COUNTS_PER_DEGREE vezes 45.

> Recordem-se de que setTargetPosition() requer um número inteiro como seu parâmetro. Ao definir armPosition, lembre-se de adicionar a linha (int) na frente da variável double. No entanto, você precisa ter cautela em relação a possíveis erros de arredondamento. Como COUNTS_PER_MM faz parte de uma equação, é recomendável converter para um número inteiro após encontrar o resultado da equação.
>
> ```
> armPosition = (int)(COUNTS_PER_DEGREE * 45);
> ```

```
while (opModeIsActive()) {


   if(gamepad1.dpad_up){
         armPosition = (int)(COUNTS_PER_DEGREE * 45);
         arm.setTargetPosition(83);
         arm.setMode(DcMotor.RunMode.RUN_TO_POSITION);
         arm.setPower(0.4);
                }
```

Defina a posição alvo para *armPosition*:

```
if(gamepad1.dpad_up){
     armPosition = (int)(COUNTS_PER_DEGREE * 45);
     arm.setTargetPosition(armPosition);
     arm.setMode(DcMotor.RunMode.RUN_TO_POSITION);
     arm.setPower(0.4);
            }
else if (gamepad1.dpad_down){
      arm.setTargetPosition(0);
      arm.setMode(DcMotor.RunMode.RUN_TO_POSITION);
      arm.setPower(0.4);
            }
```

> “ Poderíamos alterar para o que armPosition é igual na parte do gamepad1.dpad_down da instrução if/else if, como por exemplo:
>
> ```
> else if (gamepad1.dpad_down){
>   armPosition = (int)(COUNTS_PER_DEGREE * 0);
> ```

```
arm.setTargetPosition(armPosition);
arm.setTargetPosition(armPosition);
arm.setPower(0.4);
}
```

Neste caso, estaríamos constantemente redefinindo armPosition para atender às necessidades das posições que desejamos criar. Uma vez que, no momento, temos apenas duas posições - a posição inicial e a posição de 90 graus - não é necessário. No entanto, é uma boa prática criar uma variável em situações como essa. Se quisermos adicionar outra posição posteriormente, podemos editar facilmente a variável para atender às nossas necessidades.

Utilizando limites para controlar a faixa de um movimento

Nas seções anteriores, você trabalhou em alguns dos fundamentos para restringir o alcance de movimento de um braço. A partir dessas seções, você deveria ter a base necessária para realizar um controle básico do braço. No entanto, existem outras maneiras criativas de usar posições de encoder e limites para expandir o controle sobre o braço.

Esta seção abordará dois tipos adicionais de controle. O primeiro tipo de controle que exploraremos é a ideia de "soft limits" (limites suaves). Na seção de Adição de um Interruptor de Limite, discutimos o conceito de limites físicos de um mecanismo; no entanto, pode haver momentos em que você precisa limitar o alcance de movimento de um braço sem instalar um limite físico. Para fazer isso, pode-se usar código baseado em posição para criar um intervalo para o braço.

Depois de ter uma ideia básica de como criar limites suaves, exploraremos como usar um interruptor de limite (como um sensor de toque) para redefinir o alcance de movimento. Esse tipo de controle reduz o risco de ficar preso fora do intervalo pretendido, o que pode afetar o comportamento esperado do seu robô.

Para definir os limites suaves, usaremos alguma lógica básica que estabelecemos em seções anteriores, com algumas alterações editadas. Comece com um Op Mode básico e adicione as variáveis constantes da seção anterior (calculando a posição alvo).

```
@TeleOp

public class Basic extends LinearOpMode {
    private DcMotor  arm;

    static final double     COUNTS_PER_MOTOR_REV    = 288;
    static final double     GEAR_REDUCTION    = 2.7778;
    static final double     COUNTS_PER_GEAR_REV    = COUNTS_PER_MOTOR_REV * GEAR_REDUCTION;
    static final double     COUNTS_PER_DEGREE    = COUNTS_PER_GEAR_REV/360;

    @Override
    public void runOpMode() {
        arm = hardwareMap.get(DcMotor.class, "arm");



        waitForStart();

        while (opModeIsActive()) {
            telemetry.addData("Status", "Running");
            telemetry.update();

        }
    }
}
```

Depois, nós precisamos criar nossos limites superior e inferior. Portanto, crie duas novas variáveis (int), uma chamada de **minPosition** e a outra de **maxPosition**. Adicione ambos na parte de inicialização do Op Mode, acima do waitForStart();.

```
public void runOpMode() {
        arm = hardwareMap.get(DcMotor.class, "arm");


        int minPostion;
        int maxPosition;
        waitForStart();
```

Agora nós queremos que a **minPosition** defina a posição inicial, e que a **maxPosition** defina a posição de 90°. Defina a **minPosition** igual a 0 e defina **maxPosition** igual a **COUNTS_PER_DEGREE** vezes 45.

> Lembre-se de que é necessário converter os tipos primitivos

```
int minPostion = 0;
int maxPosition = (int)(COUNTS_PER_DEGREE *45);
```

Uma lógica de if/else if precisa ser adicionada no controle do braço, para isso nós podemos usar a mesma lógica que utilizamos em noções básicas da programação de um braço.

```
while(opModeIsActive()){
     if(gamepad1.dpad_up){
            arm.setPower(0.5);
            }
     else if (gamepad1.dpad_down){
            arm.setPower(-0.5);
            }
     else {
            arm.setPower(0);
            }
     }
```

Para definir o limite, precisamos editar nossa instrução if/else if para que os limites sejam incorporados. Se DpadUp for selecionado e a posição do braço for menor que a maxPosition, então o braço se moverá para a maxPosition. Se DpadDown for selecionado e a posição do braço for maior que a minPosition, então o braço se moverá em direção à minPosition.

```
while (opModeIsActive()) {
    if (gamepad1.dpad_up && arm.getCurrentPosition() < maxPosition) {
            arm.setPower(0.5);
            }
    else if (gamepad1.dpad_down && arm.getCurrentPosition() > minPosition) {
            arm.setPower(-0.5);
            }
    else {
            arm.setPower(0);
            }
    }
```

> A configuração atual do código interromperá o motor em qualquer ponto em que as condições para alimentar o motor não sejam atendidas. Dependendo de fatores como o peso do mecanismo e qualquer carga que ele esteja suportando, quando o motor para, o braço

pode cair abaixo da maxPosition. Reserve um tempo para testar o código e confirmar se ele se comporta da maneira que você espera.

Substítuindo limites

Um dos benefícios de ter um limite suave é a capacidade de ultrapassar esse limite. Como a posição zero dos ticks do encoder é determinada pela posição do braço quando o Control Hub é ligado; se não prestarmos atenção à posição do braço no momento da inicialização, o alcance de movimento do braço é afetado. Por exemplo, se precisarmos reiniciar o Control Hub enquanto o braço estiver na posição de 90 graus, a posição de 90 graus será igual a 0 ticks do encoder. Uma maneira de contornar isso é criar uma substituição para o alcance de movimento.

Existem várias maneiras diferentes de criar uma substituição desse tipo; no nosso caso, vamos usar um botão e um sensor de toque para ajudar a redefinir nosso intervalo.

Comece editando a instrução if/else if para adicionar outra condição else if. Use a linha gamepad1.a como condição. Adicione a linha arm.setPower(-0.5); como a ação correspondente.

```
while (opModeIsActive()) {
    if (gamepad1.dpad_up && arm.getCurrentPosition() < maxPosition) {
            arm.setPower(0.5);
            }
    else if (gamepad1.dpad_down && arm.getCurrentPosition() > minPosition) {
            arm.setPower(-0.5);
            }
    else if(gamepad1.a){
            arm.setPower(-0.5);
    else {
            arm.setPower(0);
            }
    }
```

Agora que temos essa alteração em vigor, quando o botão A é pressionado, o braço se moverá em direção à posição inicial. Quando o braço alcançar e pressionar o sensor de toque, queremos utilizar STOP_AND_RESET_ENCODER.

Podemos criar outra instrução if que se concentra em realizar essa parada e reinicialização quando o sensor de toque é pressionado. Como o sensor de toque relata true quando não está pressionado e false quando está, precisaremos usar o operador lógico not !.

O operador ! pode ser usado em instruções binárias condicionais quando você precisa inverter se algo é verdadeiro ou falso. Por exemplo, uma instrução if é ativada quando algo é verdadeiro, mas quando touch.getState(); relata verdadeiro, significa que não está

> pressionado. No nosso caso, queremos que esta instrução if seja ativada quando o sensor de toque está pressionado, portanto, precisamos usar o operador not.

```
if (!touch.getState()) {
          arm.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
        }
```

Portanto, se o sensor de toque retornar falso (ou estiver pressionado), o modo de execução do motor STOP_AND_RESET_ENCODER será ativado, fazendo com que o encoder do motor seja redefinido para 0 ticks.

Agora que este código está concluído, experimente testá-lo!

```
@TeleOp

public class HelloRobot_ArmControl extends LinearOpMode {
    private DcMotor arm;
    private Servo claw;
    private Gyroscope imu;
    private DcMotor leftmotor;
    private DcMotor rightmotor;
    private DigitalChannel touch;

    static final double     COUNTS_PER_MOTOR_REV    = 288;
    static final double     GEAR_REDUCTION    = 2.7778;
    static final double     COUNTS_PER_GEAR_REV    = COUNTS_PER_MOTOR_REV * GEAR_REDUCTION;
    static final double     COUNTS_PER_DEGREE    = COUNTS_PER_GEAR_REV/360;



    @Override
    public void runOpMode() {
        arm = hardwareMap.get(DcMotor.class, "arm");
        claw = hardwareMap.get(Servo.class, "claw");
        imu = hardwareMap.get(Gyroscope.class, "imu");
        leftmotor = hardwareMap.get(DcMotor.class, "leftmotor");
        rightmotor = hardwareMap.get(DcMotor.class, "rightmotor");
        touch = hardwareMap.get(DigitalChannel.class, "touch");

        int minPostion = 0;
        int maxPosition = (int)(COUNTS_PER_DEGREE *45);
```

```
        waitForStart();

        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {
            if (gamepad1.dpad_up && arm.getCurrentPosition() < maxPosition) {
                arm.setPower(0.5);
                    }
            else if (gamepad1.dpad_down && arm.getCurrentPosition() > minPosition) {
                arm.setPower(-0.5);
                    }
            else if (gamepad1.a) {
                arm.setPower(-0.5);
                    }
            else {
                arm.setPower(0);
                }
            if (!touch.getState()) {
              arm.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
                    }
        telemetry.addData("Arm Test", arm.getCurrentPosition());
        telemetry.update();
        }
    }
}
```

Revisão #2
Criado 2023-12-18 19:46:36 UTC por Enzo
Atualizado: 2023-12-19 12:10:08 UTC por Enzo