

Encoders

- [Encoders](#)
- [Codificadores dos motores REV](#)
- [Utilizando encoders](#)
 - [Programando encoders](#)

Encoders

O que é um encoder?

Um codificador é qualquer coisa (dispositivo, software, pessoa) que converte informações de um formato para outro. Alguns exemplos de codificação incluem:

- Um transdutor, como um alto-falante, que converte um sinal elétrico em ondas sonoras.
- Software que codifica um arquivo de áudio em um formato mp3 para reduzir o tamanho do arquivo.
- Um estenógrafo (repórter de tribunal) que registra o diálogo da sala do tribunal e o converte em um registro escrito. Esta seção trata de codificadores rotativos, que são dispositivos eletromecânicos que convertem a posição angular de um eixo, como o de um motor, em um sinal eletrônico. Esses sinais podem ser alimentados em um microcontrolador, que controla todas as funções do robô, e então utilizados para fornecer dados do mundo real para tomar decisões de programação mais precisas. Existem dois tipos principais de codificadores: absolutos e relativos.

Os **codificadores absolutos** retornam o ângulo real da rotação (por exemplo, 30°). Eles mantêm as informações de posição mesmo se a energia for removida, e os dados de posição estão imediatamente disponíveis quando a energia é restabelecida, sem necessidade de rotação para ler o ângulo atual. A relação entre o valor do codificador e o eixo do motor é estabelecida durante a montagem e permanecerá sempre a mesma. Comumente, esses codificadores usam um disco com um padrão especialmente impresso, que é lido e convertido em um ângulo conhecido. Geralmente, os codificadores absolutos são mais fáceis de usar na programação, mas são mais complicados de fabricar, sendo assim, são maiores ou mais caros.

Os **codificadores relativos**, também conhecidos como codificadores incrementais, fornecem informações sobre o movimento do eixo (por exemplo, para a frente a 5 RPM) e só fornecem dados enquanto o eixo está girando. Uma maneira de lembrar isso é que os codificadores relativos retornam informações sobre a mudança incremental do eixo de saída do motor. Eles fornecem apenas pulsos à medida que o motor gira, e a interpretação desses pulsos em informações úteis deve ser feita externamente. Um codificador relativo não sabe em que posição está ao iniciar, mas é possível criar um programa de calibração que deve ser executado a cada inicialização para obter um ponto de referência para calcular um ângulo.

Os codificadores medem uma mudança no mundo real (rotação do eixo) e a convertem em um sinal elétrico. Duas formas comuns de fazer isso são usando feedback óptico ou magnético:

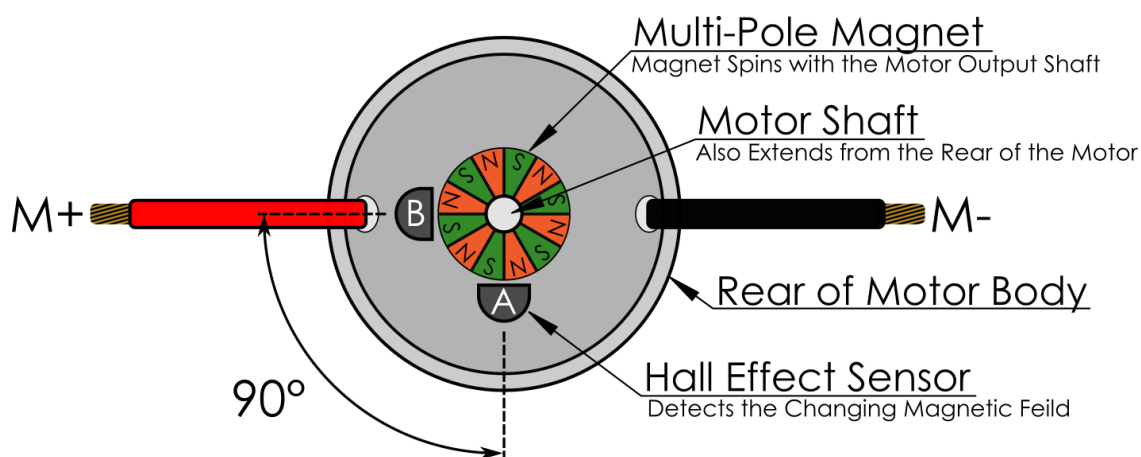
Codificadores ópticos possuem um disco com uma série de fendas ou um padrão reflexivo ao redor da borda, que está conectado ao eixo do motor. Uma luz incide sobre ou através do disco,

onde a luz pode passar ou refletir em um fotodiodo (dispositivo que produz um sinal elétrico quando a luz incide sobre ele). Esses sensores podem ser muito leves e compactos, mas podem ser muito sensíveis a qualquer coisa que possa interferir na luz que chega ao fotodiodo. Impressões digitais em um disco reflexivo ou poeira de um ambiente sujo podem interferir.

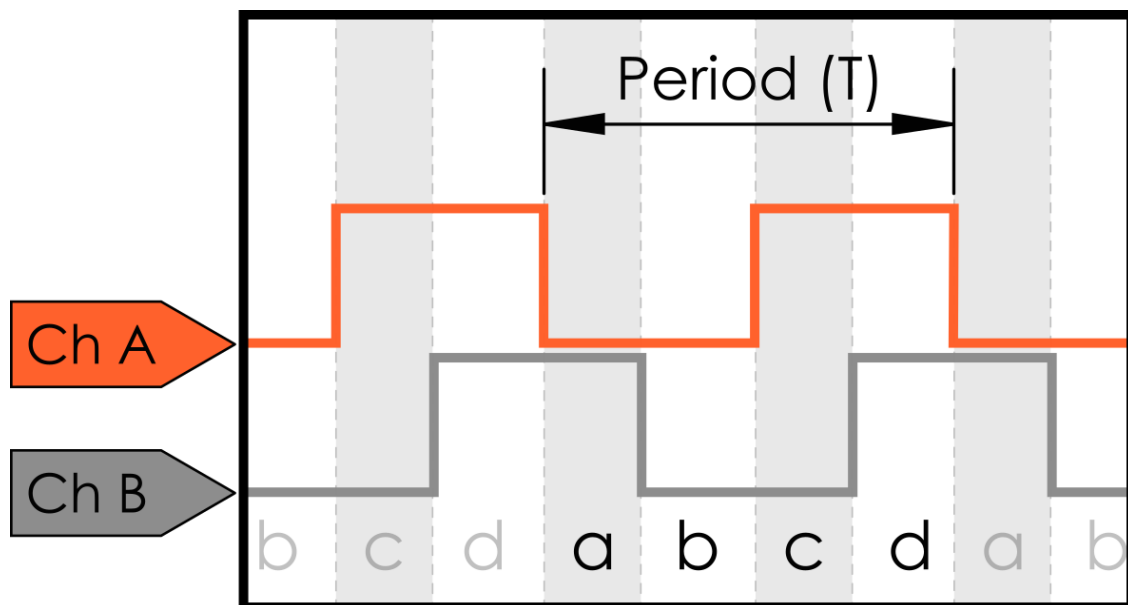
Codificadores magnéticos têm um ímã conectado ao eixo de um motor e utilizam sensores de efeito Hall para detectar o campo magnético em mudança à medida que o eixo gira. Os codificadores magnéticos conseguem operar em ambientes adversos ou sujos.

Codificadores de quadratura magnética

Um codificador magnético de quadratura de 12 polos está instalado na parte traseira tanto do Motor HD Hex quanto do Motor Core Hex. O eixo de saída do motor se estende da parte traseira do gabinete do motor, e um ímã permanente de múltiplos polos está preso ao eixo. Há dois sensores de efeito Hall, marcados como 'A' e 'B', montados ao lado do ímã em um ângulo de 90° um em relação ao outro. À medida que cada um dos 12 polos passa sobre um dos sensores de efeito Hall, cria-se uma alteração no campo magnético, fazendo com que o sensor gere um sinal de tensão mensurável.



Os codificadores de quadratura são um tipo específico de codificador relativo que possui quatro estados de saída diferentes. Se a raiz "quad-" significa quatro, mas há apenas dois sensores neste codificador, de onde vem o nome? As saídas dos dois sensores de efeito Hall são chamadas de "Canal A" e "Canal B" respectivamente; um exemplo da saída é mostrado abaixo. Em um único período (T), definido como a duração de tempo de um ciclo completo em um padrão de repetição, o diagrama de temporização tem quatro estados distintos (veja a, b, c e d abaixo), daí o nome codificador de quadratura.



O deslocamento de Canal A para Canal B ocorre porque os sensores estão deslocados um do outro por 90°. Conforme o motor gira, um sensor perceberá a mudança antes do outro. Quando o eixo do motor gira no sentido horário (CW), o Canal A irá adiantar-se (a borda subirá antes) do Canal B. Quando o motor gira no sentido anti-horário (CCW), o Canal A irá atrasar-se (subir depois) do Canal B. Se houvesse apenas um sensor, ainda seria possível medir o número de rotações, mas não detectar a direção do motor.

Nos motores HD Hex e Core Hex, o Canal A precede o Canal B quando uma tensão positiva é aplicada ao terminal M+. No entanto, há momentos em que isso pode não ser verdadeiro na prática. Diferentes caixas de redução ou a troca física dos fios do Canal A e Canal B no controlador podem inverter a relação entre os canais. Lembre-se disso ao programar e solucionar problemas no seu robô.

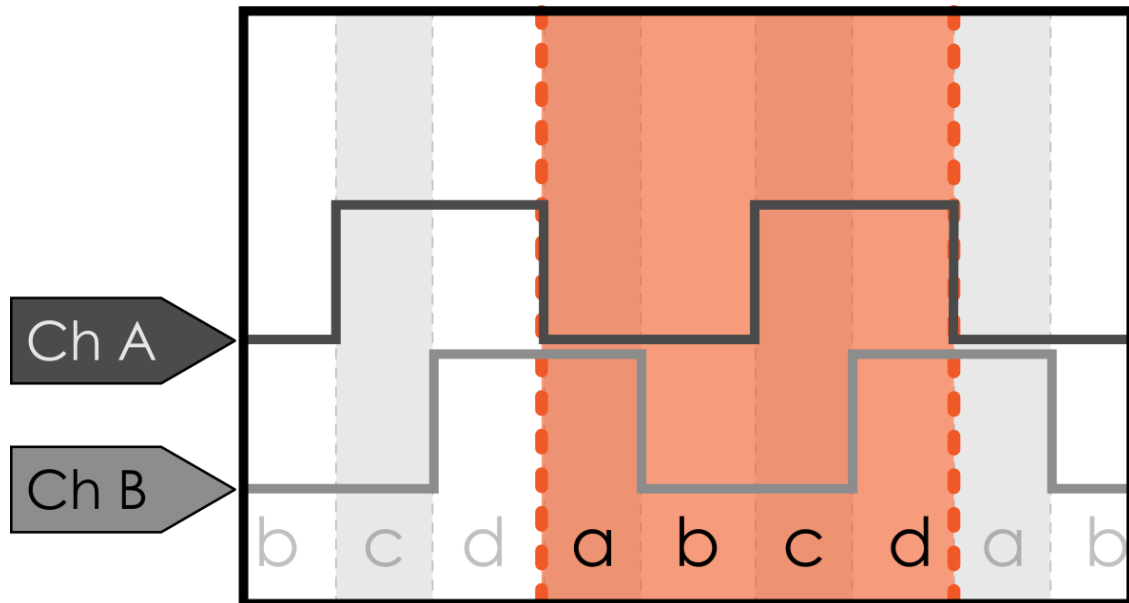
Quando o encoder está sendo lido por um microcontrolador, os dois sinais são comparados para produzir um pulso de contagem para cima ou um pulso de contagem para baixo. Esses pulsos são contados como passos para frente (CW) ou para trás (CCW). Utilizando as especificações do encoder em uso, uma contagem pode ser convertida em graus. Essas informações podem ser usadas para posicionar um braço robótico em um ângulo específico ou instruir um robô a percorrer uma determinada distância. Tanto o Control Hub quanto o Expansion Hub se comunicam com um microcontrolador por meio das portas do encoder.

Definições de especificações técnicas do codificador

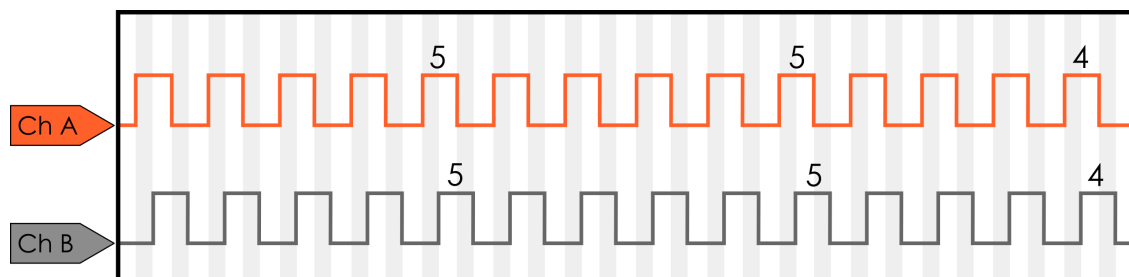
Existe alguma divergência terminológica entre fornecedores de encoders. Este documento define um conjunto de termos mais amplamente aceitos, no entanto, esteja ciente de que, ao comparar as especificações dos encoders de diferentes fornecedores, os termos podem

variar em significado.

Cada vez que a saída passa por todas as quatro combinações distintas de sinais de saída, isso é chamado de ciclo (veja a, b, c e d abaixo). Os encoders têm ciclos diferentes por revolução (CPR) com base no número de polos no ímã utilizado. O CPR é a quantidade de ciclos gerados para uma rotação completa do eixo do encoder.



Um exemplo de saída de uma rotação completa de um encoder de 14 CPR é mostrado na figura abaixo. Um encoder de rotação de 14 CPR também pode ser chamado de ter 14 transições em canal A. Os encoders são montados no eixo do motor, não no eixo de saída da caixa de redução, então, para um motor com uma caixa de redução acoplada, isso representa menos de uma rotação completa do eixo de saída.



Uma razão para usar CPR (Cycles per Revolution) para definir um encoder, em vez do comumente usado PPR (Pulsos por Revolução), é que, ao decodificar o sinal do encoder no microcontrolador, é possível realizar uma decodificação de 1x, 2x ou 4x. Para a decodificação 1x, o microcontrolador contaria apenas o sinal de subida em um único canal, enquanto para a decodificação 4x, cada borda de subida ou descida em ambos os canais é contada como um "count". Embora a decodificação 4x seja um dos métodos mais comuns, porque é baseada em como a eletrônica decodifica o sinal do encoder e não no hardware do encoder em si, não é um método ideal para definir as especificações do hardware do encoder.

Se assumirmos a decodificação 4x, onde cada ciclo é interpretado, o microcontrolador pode ler as quatro saídas distintas (a, b, c e d) como passos individuais. Assim, para cada CPR, o controlador

pode ler quatro contagens. Para calcular o número de contagens por rotação do eixo do encoder:

$$\text{CountsPerRotation(saída)} = \text{CPR (ciclos por rotação)} \times 4$$

O número real de ciclos por rotação do eixo de saída do motor depende da caixa de redução que está acoplada.

$$\text{CountsPerRotation(saída)} = \text{CPR (ciclos por rotação)} \times 4 \times \text{redução}$$

Isso pode ser calculado em graus por contagem. Supondo que não haja redução adicional adicionada à última etapa da saída do motor (ou seja, acionamento direto), o número de graus por contagem é calculado como:

$$\text{GrausPorContagem} = 360 / \text{CPR}$$

A terminologia de quadratura pode ser muito confusa! Alguns encoders podem relatar "pulsos por revolução". Um pulso pode equivaler a 1 ou 4 contagens. Outros encoders podem relatar "ciclos por revolução", o que, de forma confusa, tem o mesmo acrônimo de contagens por revolução. A melhor maneira de verificar é conectar o encoder ao REV Hub, girá-lo uma revolução completa e então conferir o que ele reporta.

Codificadores dos motores

REV

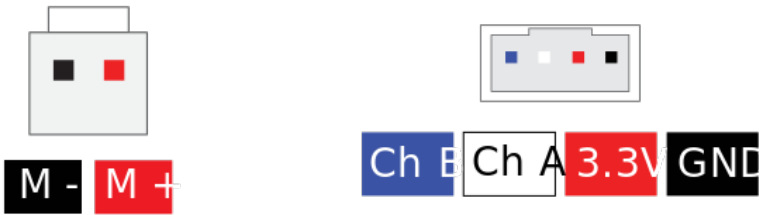
Os Motores REV Robotics HD Hex (REV-41-1291) e os Motores Core Hex (REV-41-1300) vêm com um codificador magnético de quadratura já instalado e um cabo apropriado para conectar a saída do encoder ao REV Robotics Control Hub (REV-31-1595) ou ao Expansion Hub (REV-31-1153). Consulte a Tabela 1 e a Tabela 2 para obter detalhes relevantes sobre o encoder.

Especificações do codificador do motor Core Hex

Especificações	Detalhes
Redução do motor Core Hex	72:1
Velocidade Livre (RPM)	125
CPR do eixo	4 (1 subida do canal A)
CPR fora do eixo	288 (72 subidas do canal A)

Pinagem

O Motor Core Hex utiliza um conector JST-VH de 2 pinos para a alimentação do motor e um conector JST-PH de 4 pinos para o feedback do sensor do encoder incorporado. Para obter mais informações sobre o uso dos cabos e conectores incluídos com o Motor Core Hex, consulte a documentação do [Sistema de Controle REV - Cabos e Conectores](#). A imagem abaixo apresenta a pinagem para a alimentação do motor e o encoder.

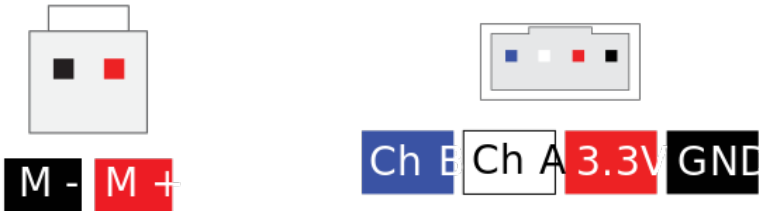


Especificações do codificador do HD Hex

Especificações	Tipo do motor		
Redução HD Hex	Apenas o motor	40:1	20:1
Velocidade Livre (RPM)	6000	150	300
CPR do eixo do codificador	28 (7 subidas no canal A)	28 (7 subidas no canal A)	28 (7 subidas no canal A)
CPR no eixo externo	28 (7 subidas no canal A)	1120 (280 subidas no canal A)	560 (140 subidas no canal A)

Pinagem

The HD Hex Motor utiliza um conector JST-VH de 2 pinos para a alimentação do motor e um conector JST-PH de 4 pinos para o feedback do sensor do encoder incorporado. Para obter mais informações sobre o uso dos cabos e conectores incluídos com o HD Hex Motor, consulte a documentação do [Sistema de Controle REV - Cabos e Conectores](#). A imagem abaixo apresenta a pinagem para a alimentação do motor e o encoder.



Utilizando encoders

Este capítulo tem como objetivo ensinar como programar um encoder para o seu robô.

Programando encoders

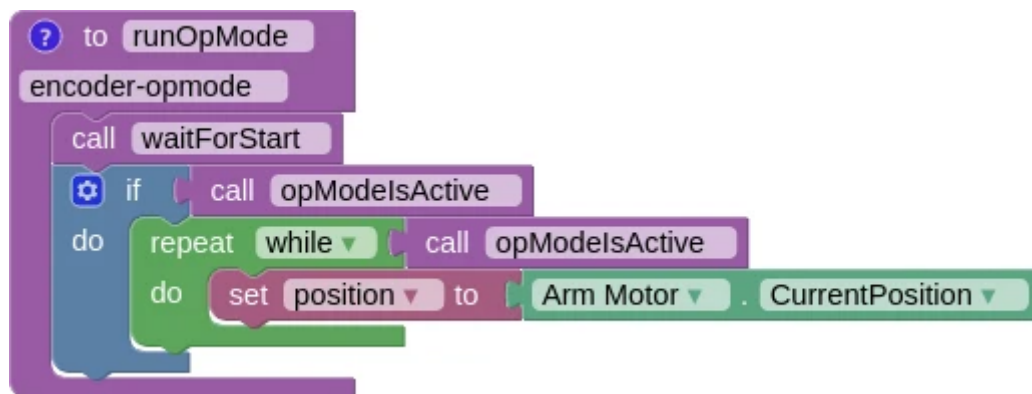
Leitura de encoders

No software do FTC, encoders de quadratura e motores são acessados com o mesmo objeto de motor. O que isso significa é que podemos acessar a posição de um encoder da seguinte maneira:

Java

```
int position = motor.getCurrentPosition();
```

Blocks

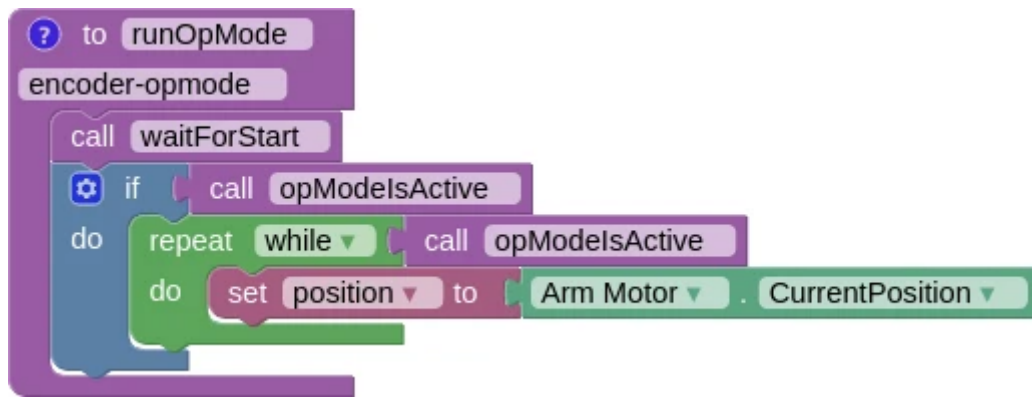


Embora seja conveniente usar o encoder de motor embutido, isso pode se tornar confuso ao usar encoders externos. Para usar encoders externos, você deve usar o objeto do motor associado à porta. Por exemplo, se houver um motor na porta 1 chamado "Arm Motor" e um encoder externo conectado à porta de encoder 1, você deve fazer o seguinte para obter a posição do encoder:

Java

```
DcMotor motor = hardwareMap.dcMotor.get("Arm Motor");  
double position = motor.getCurrentPosition();
```

Blocks

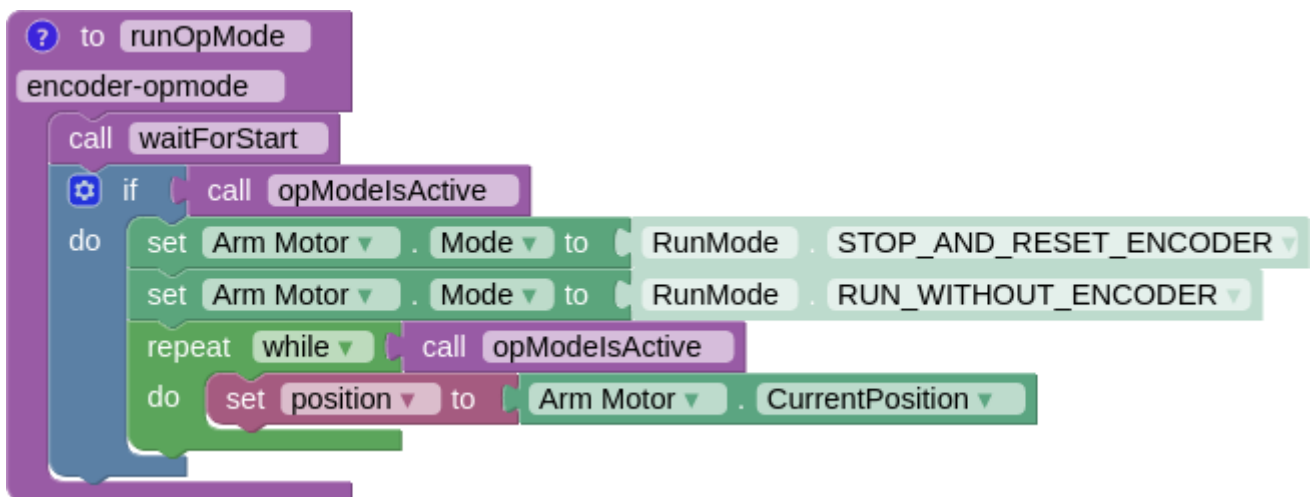


Ótimo! Agora temos a posição relativa do nosso encoder, reportada pelo número de "contagens" que ele se afastou do que considera ser o zero. No entanto, muitas vezes é conveniente fazer o encoder começar em zero no início do OpMode. Embora tecnicamente isso não mude nada, pode ajudar na depuração e simplificar o código futuro. Para fazer isso, podemos adicionar uma chamada para resetar os encoders antes de lê-los.

Java

```
DcMotor motor = hardwareMap.dcMotor.get("Arm Motor");  
motor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER); // Reset the motor encoder  
motor.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER); // Turn the motor back on when we are done  
int position = motor.getCurrentPosition();
```

Blocks



Como observação, **RUN_WITHOUT_ENCODER** não desativa o encoder. Ele apenas informa ao SDK para não usar o encoder do motor para o controle de velocidade embutido. Vamos explicar o que isso significa em uma seção posterior, mas por enquanto, saiba que ele liga o motor novamente para que possamos usá-lo após o encoder ser resetado.

Agora, temos nossa posição (em contagens) relativa ao ângulo inicial do encoder. Podemos criar um programa rápido para ver isso em ação. Aqui, usamos um encoder de motor conectado a uma porta chamada "Arm Motor" na configuração de hardware.

Java

```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;
@TeleOp
public class EncoderOpmode extends LinearOpMode {
    @Override
    public void runOpMode() throws InterruptedException {
        // Find a motor in the hardware map named "Arm Motor"
        DcMotor motor = hardwareMap.dcMotor.get("Arm Motor");

        // Reset the motor encoder so that it reads zero ticks
        motor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

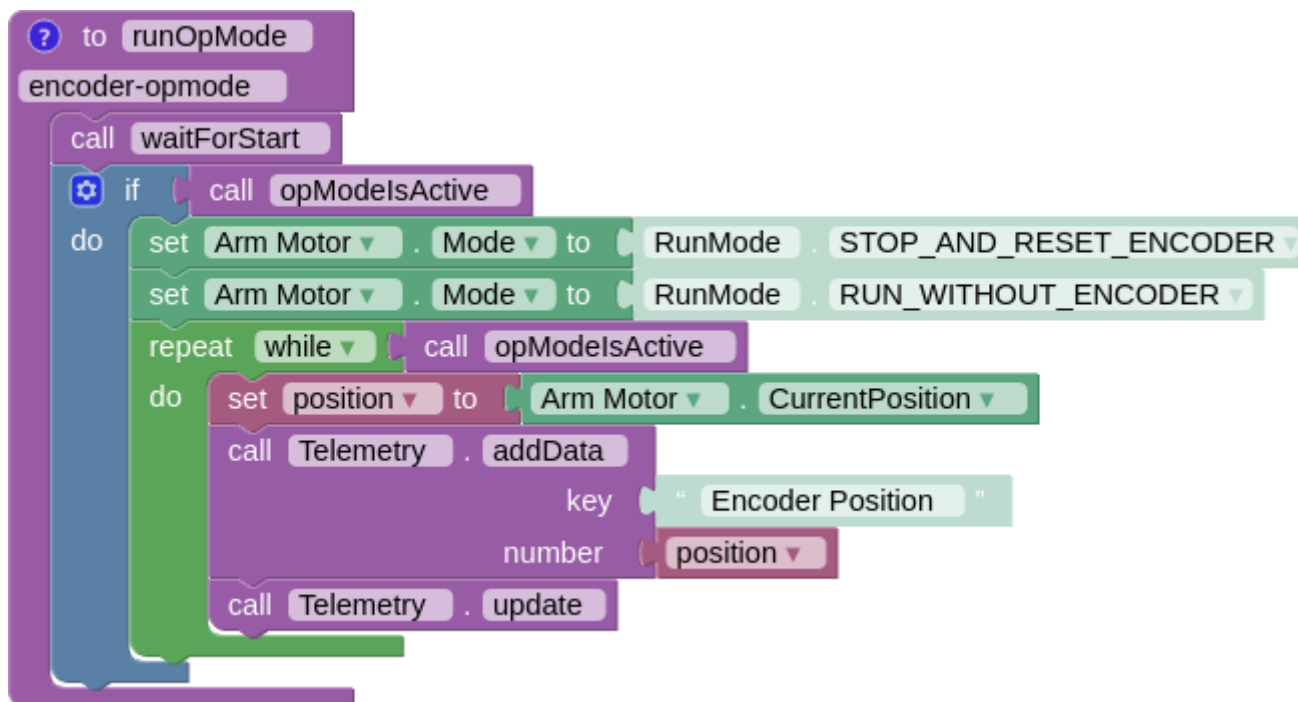
        // Turn the motor back on, required if you use STOP_AND_RESET_ENCODER
        motor.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);

        waitForStart();

        while (opModelsActive()) {
            // Get the current position of the motor
            int position = motor.getCurrentPosition();

            // Show the position of the motor on telemetry
            telemetry.addData("Encoder Position", position);
            telemetry.update();
        }
    }
}
```

Blocks



Se você executar o OpMode acima e girar o encoder, deverá ver os valores mudarem conforme você se move. Se girar o eixo de volta para onde ele começou, verá o número retornar (muito próximo) de zero. Como exercício, gire o eixo uma revolução completa (360 graus) e anote o número.

Há mais uma coisa que podemos fazer com os encoders. Embora saber o número de contagens que algo se moveu seja útil, muitas vezes será necessário um número diferente, como o número de revoluções que o encoder fez ou o ângulo em que ele está. Para determinar esses valores, precisamos de uma constante: as Contagens Por Revolução (CPR) do encoder. Para encoders externos, esse número geralmente é fornecido na ficha técnica. Para motores, normalmente estará na página do produto, embora alguns motores (notavelmente o REV Ultraplanetary Gearbox) não forneçam isso de forma clara.

Você pode calcular as Contagens Por Revolução de um motor pegando o Counts Per Revolution do motor base e multiplicando-o pela razão de redução da caixa de engrenagens. Tenha cuidado para usar a razão de redução real da caixa de engrenagens ao fazer isso! Por exemplo, um motor Ultraplanetary 5:1 teria uma contagem por revolução de $28 * 5.23 = 146.44$, pois o motor base tem 28 Counts Per Revolution e a caixa de engrenagens 5:1 tem uma razão de redução de 5.23:1. Lembre-se, ao usar duas caixas de engrenagens uma sobre a outra, você deve multiplicar as razões de redução entre si.

No exemplo a seguir, dividimos a posição do codificador pelo número de contagens por revolução para obter o número de revoluções que o codificador realizou. Você deve substituir [Your Counts Per Revolution Here] pelo número de contagens por revolução do seu motor, que pode ser encontrado na página do produto ou calculado usando a dica acima.

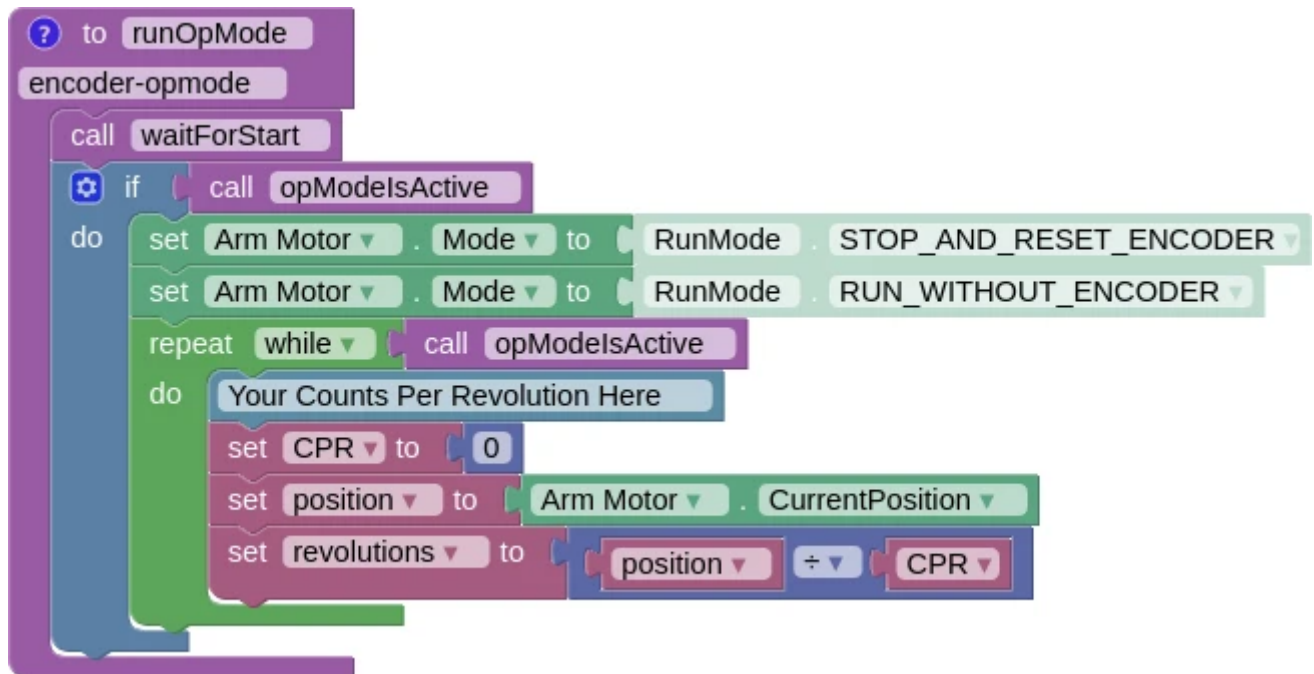
Java

```
double CPR = [Your Counts Per Revolution Here];
```

```
int position = motor.getCurrentPosition();
```

```
double revolutions = position/CPR;
```

Blocks



Há um número a mais que podemos obter: o ângulo do eixo. Calcular esse número é muito simples. Podemos multiplicar o número de rotações por 360 (já que há 360 graus em uma revolução). Você pode notar que esse número pode ultrapassar 360 à medida que o eixo gira várias vezes. Assim, introduzimos o `angleNormalized`, que estará sempre entre 0 e 360.

Java

```
double CPR = [Your Counts Per Revolution Here];
```

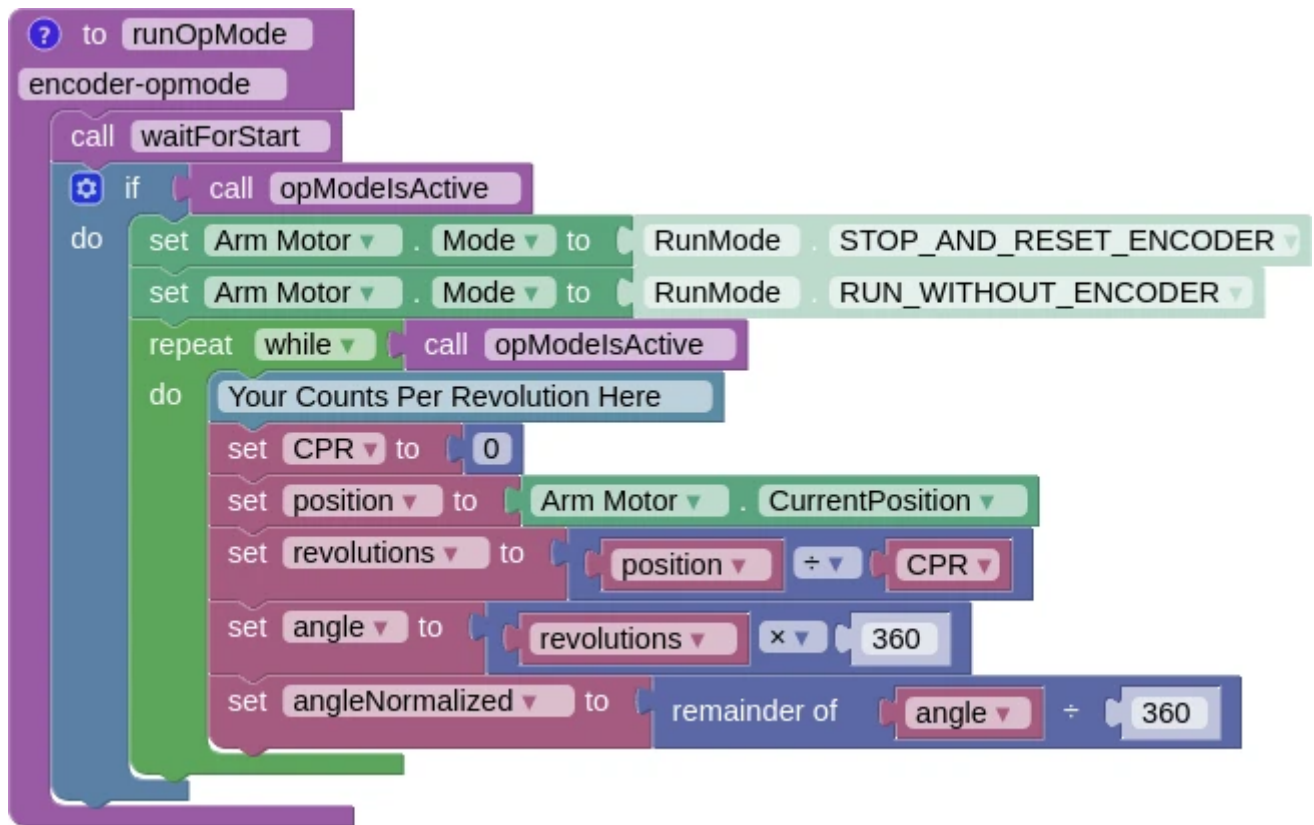
```
int position = motor.getCurrentPosition();
```

```
double revolutions = position/CPR;
```

```
double angle = revolutions * 360;
```

```
double angleNormalized = angle % 360;
```

Blocks



Juntando tudo, obtemos o seguinte programa de teste.

Java

```

package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;

@TeleOp
public class EncoderOpmode extends LinearOpMode {
    @Override
    public void runOpMode() throws InterruptedException {
        // Find a motor in the hardware map named "Arm Motor"
        DcMotor motor = hardwareMap.dcMotor.get("Arm Motor");

        // Reset the motor encoder so that it reads zero ticks
        motor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

        // Turn the motor back on, required if you use STOP_AND_RESET_ENCODER
        motor.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
    }
}

```

```
waitForStart();
```

```
while (opModelsActive()) {
```

```
    double CPR = [Your Counts Per Revolution Here];
```

```
    // Get the current position of the motor
```

```
    int position = motor.getCurrentPosition();
```

```
    double revolutions = position/CPR;
```

```
    double angle = revolutions * 360;
```

```
    double angleNormalized = angle % 360;
```

```
    // Show the position of the motor on telemetry
```

```
    telemetry.addData("Encoder Position", position);
```

```
    telemetry.addData("Encoder Revolutions", revolutions);
```

```
    telemetry.addData("Encoder Angle (Degrees)", angle);
```

```
    telemetry.addData("Encoder Angle - Normalized (Degrees)", angleNormalized);
```

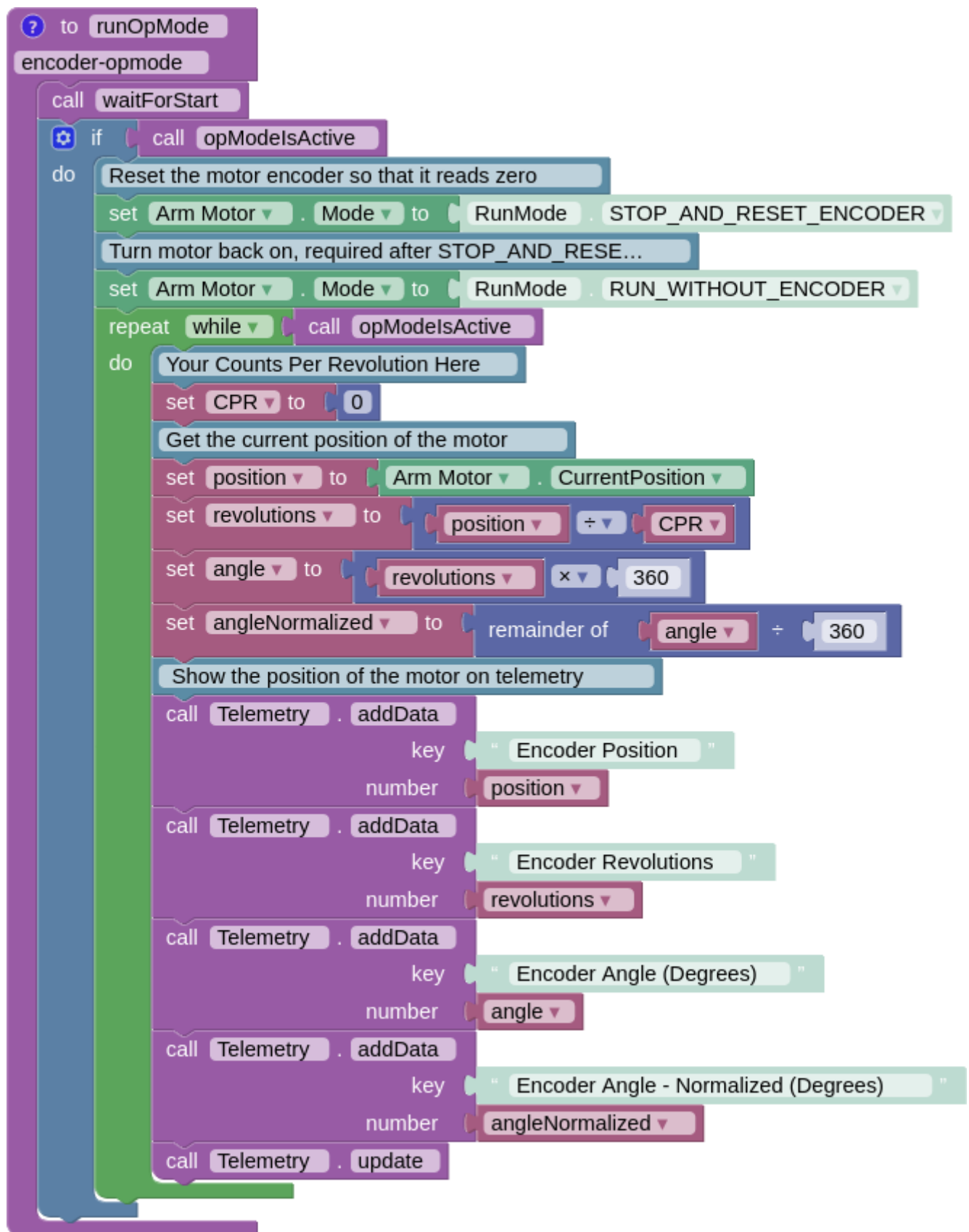
```
    telemetry.update();
```

```
}
```

```
}
```

```
}
```

Blocks



Rastreando rodas e outros atuadores lineares

Até este ponto, temos trabalhado principalmente com motores girando algo. No entanto, muitos mecanismos no FTC são lineares, e pode ser desejável medir esses mecanismos em uma unidade linear também. Felizmente, isso é bem simples. Tudo o que precisamos saber é o diâmetro do

objeto que estamos medindo.

Tenha cuidado ao selecionar seu diâmetro. Para rodas, é o diâmetro externo da roda, mas para carretéis, é o diâmetro interno do carretel, onde o cordão repousa. Para correntes e correias, é o "diâmetro de passo" da engrenagem ou polia.

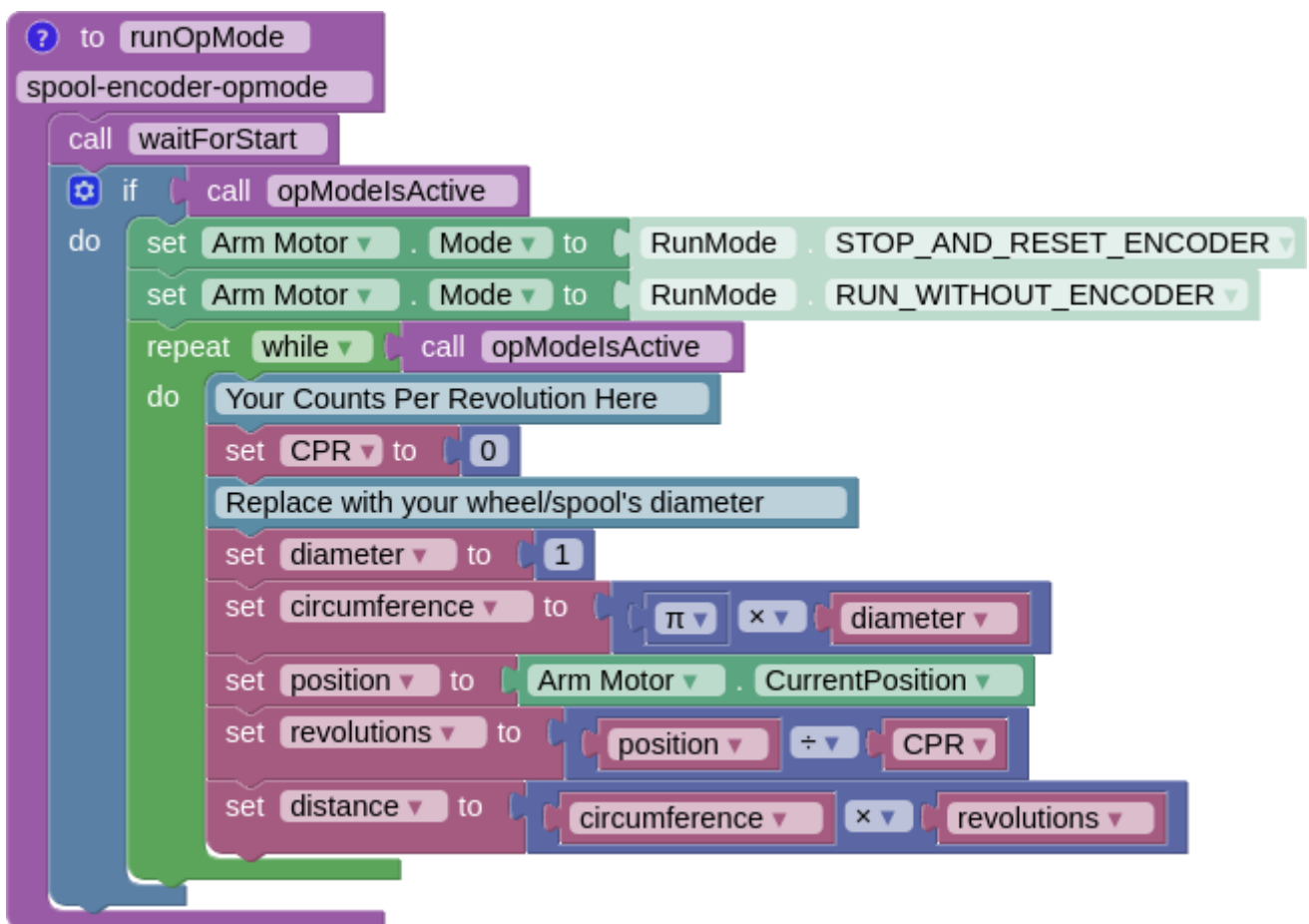
A partir daí, podemos calcular a circunferência (o comprimento do arco do círculo, ou a distância que a roda/carretel percorrerá em uma rotação).

Java

```
double diameter = 1.0; // Replace with your wheel/spool's diameter
double circumference = Math.PI * diameter;

double distance = circumference * revolutions;
```

Blocks



Unidades são muito importantes ao lidar com programação FTC, então certifique-se de que elas sejam sempre consistentes! Quaisquer unidades que você usar para o diâmetro serão as unidades para a distância calculada. Portanto, se você medir o diâmetro em polegadas, a

distância reportada também será em polegadas.

Juntando tudo isso com o nosso programa de teste anterior, obtemos:

Java

```
package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;
@TeleOp
public class SpoolEncoderOpmode extends LinearOpMode {
    @Override
    public void runOpMode() throws InterruptedException {
        // Find a motor in the hardware map named "Arm Motor"
        DcMotor motor = hardwareMap.dcMotor.get("Arm Motor");

        // Reset the motor encoder so that it reads zero ticks
        motor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

        // Turn the motor back on, required if you use STOP_AND_RESET_ENCODER
        motor.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);

        waitForStart();

        while (opModeIsActive()) {
            double CPR = [Your Counts Per Revolution Here];

            double diameter = 1.0; // Replace with your wheel/spool's diameter
            double circumference = Math.PI * diameter;

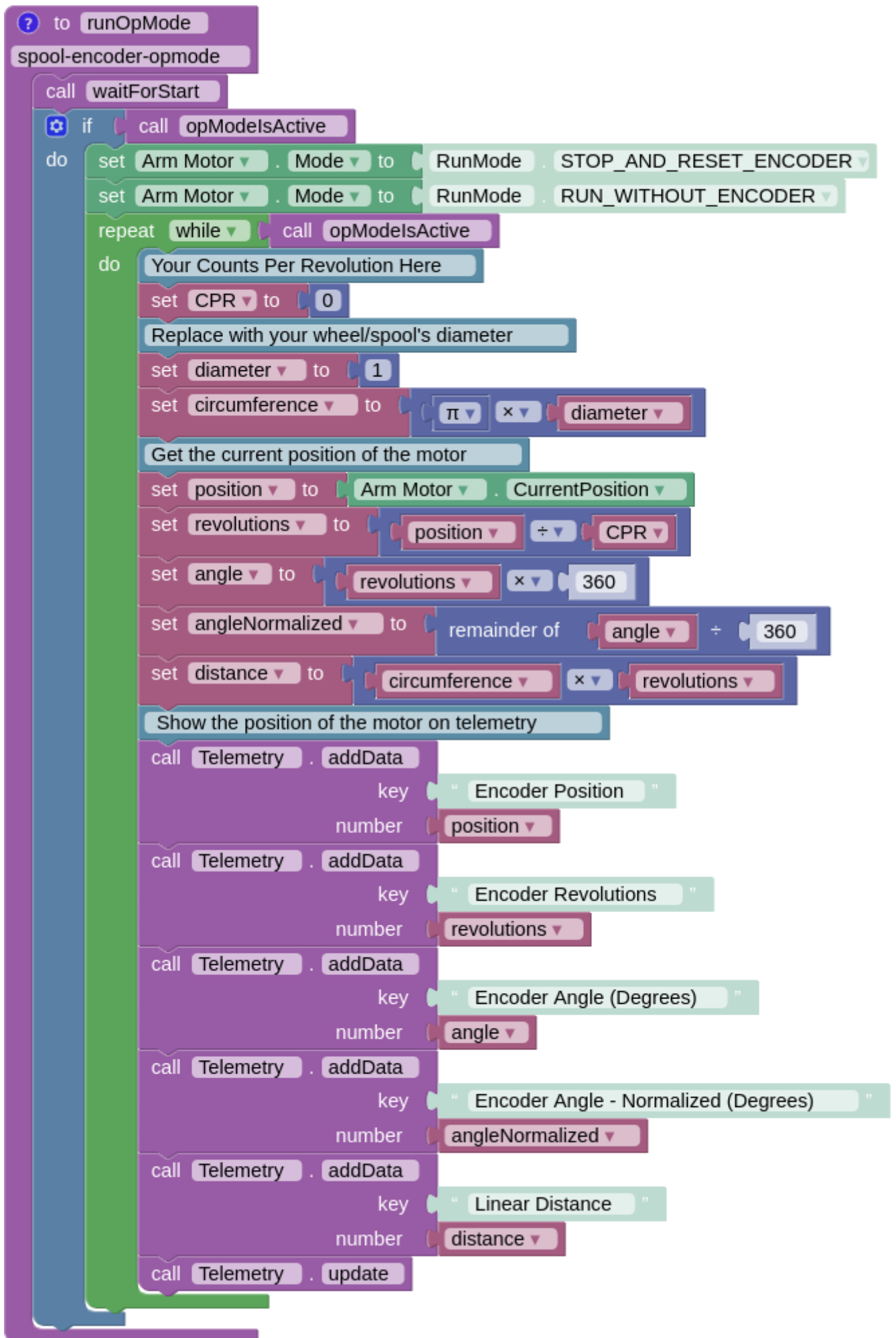
            // Get the current position of the motor
            int position = motor.getCurrentPosition();
            double revolutions = position/CPR;

            double angle = revolutions * 360;
            double angleNormalized = angle % 360;

            double distance = circumference * revolutions;
```

```
//Show the position of the motor on telemetry
telemetry.addData("Encoder Position", position);
telemetry.addData("Encoder Revolutions", revolutions);
telemetry.addData("Encoder Angle (Degrees)", angle);
telemetry.addData("Encoder Angle - Normalized (Degrees)", angleNormalized);
telemetry.addData("Linear Distance", distance);
telemetry.update();
}
}
}
```

Blocks



Operando motores com encoders

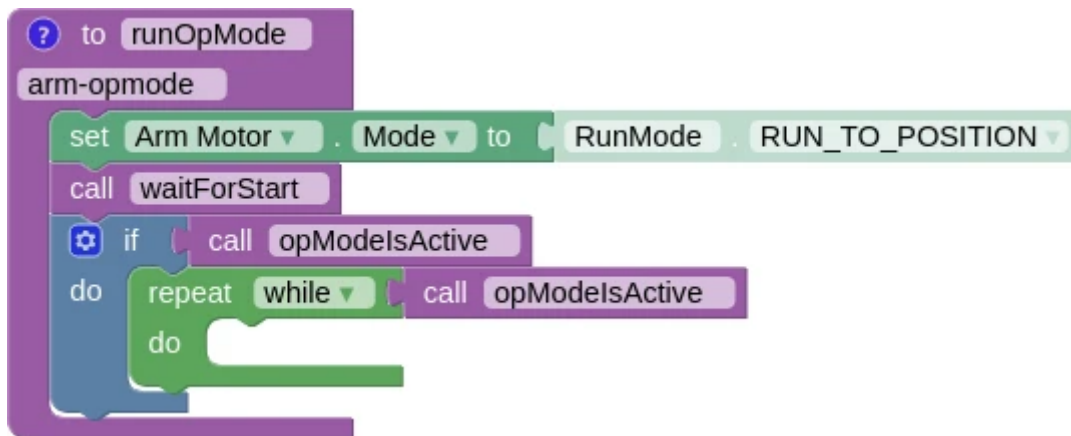
Aprendemos como ler os valores do codificador, mas como definimos onde queremos ir e dizemos ao motor para ir até lá?

Anteriormente, aprendemos sobre o modo RUN_WITHOUT_ENCODER para o motor. Podemos usar outro modo de motor, RUN_TO_POSITION, para dizer ao motor para rodar até uma posição específica em contagens, assim:

Java

```
DcMotor motor = hardwareMap.dcmotor.get("Arm Motor");  
motor.setMode(DcMotor.RunMode.RUN_TO_POSITION); // Tells the motor to run to the specific position
```

Blocks



Você pode descobrir mais sobre os modos de execução na página oficial da documentação da REV Robotics.

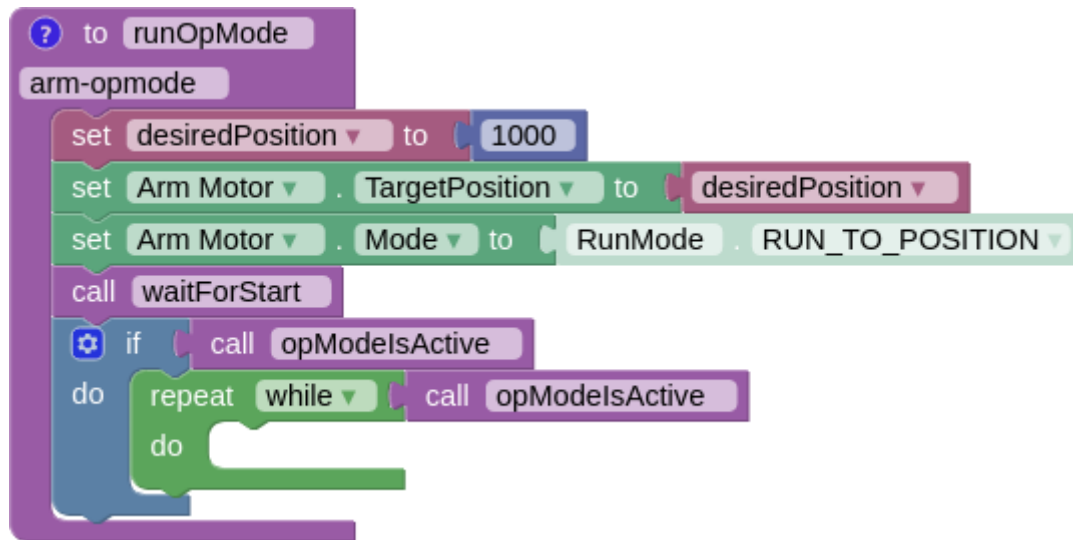
No entanto, antes de dizer ao motor para ir até uma posição, precisamos informar ao motor qual posição ele deve atingir. Observe que esse valor deve ser um número inteiro. Vamos modificar o código acima para fazer isso.

Definir o motor para o modo RUN_TO_POSITION antes de definir a posição alvo gerará um erro. Tenha cuidado para não fazer isso!

Java

```
DcMotor motor = hardwareMap.dcmotor.get("Arm Motor");  
int desiredPosition = 1000; // The position (in ticks) that you want the motor to move to  
motor.setTargetPosition(desiredPosition); // Tells the motor that the position it should go to is desiredPosition  
motor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
```

Blocks



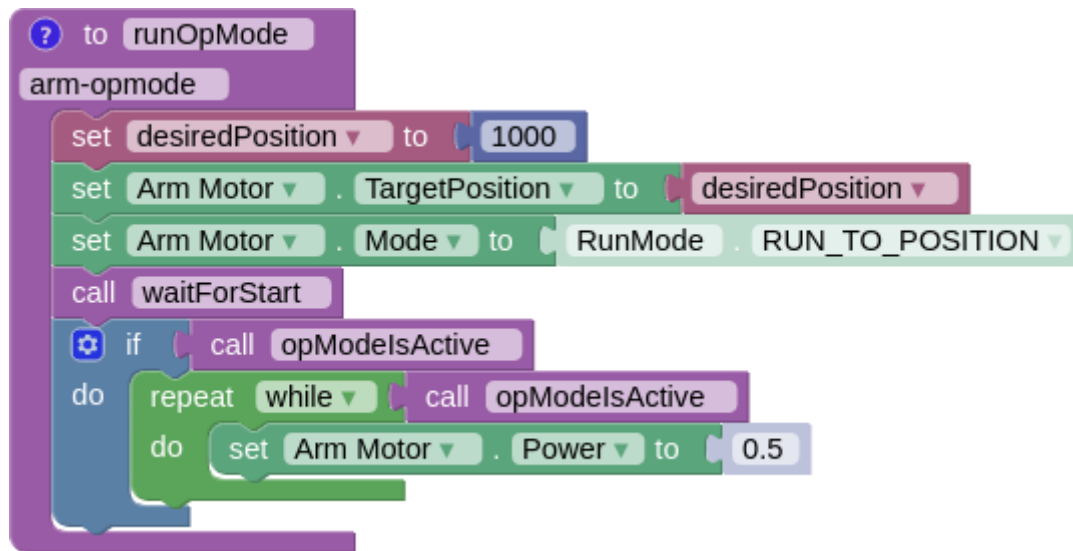
Este código diz ao motor para se mover até 1000 contagens, usando um loop PID para controlar a posição do motor. Você pode ler mais sobre loops PID [aqui](#).

Podemos limitar a velocidade com a qual o motor corre usando o seguinte código:

Java

```
DcMotor motor = hardwareMap.dcmotor.get("Arm Motor");  
int desiredPosition = 1000; // The position (in ticks) that you want the motor to move to  
motor.setTargetPosition(desiredPosition); // Tells the motor that the position it should go to is desiredPosition  
motor.setMode(DcMotor.RunMode.RUN_TO_POSITION);  
motor.setPower(0.5); // Sets the maximum power that the motor can go at
```

Blocks



Agora, vamos usar essas informações para controlar um braço em um OpMode.

Java

```
package org.firstinspires.ftc.teamcode.Tests;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;

@TeleOp
public class ArmOpMode extends LinearOpMode {
    @Override
    public void runOpMode() throws InterruptedException {
        // Position of the arm when it's lifted
        int armUpPosition = 1000;

        // Position of the arm when it's down
        int armDownPosition = 0;

        // Find a motor in the hardware map named "Arm Motor"
        DcMotor armMotor = hardwareMap.dcMotor.get("Arm Motor");

        // Reset the motor encoder so that it reads zero ticks
        armMotor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

        // Sets the starting position of the arm to the down position
        armMotor.setTargetPosition(armDownPosition);
```



```

armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);

waitForStart();

while (opModelsActive()) {
    // If the A button is pressed, raise the arm
    if (gamepad1.a) {
        armMotor.setTargetPosition(armUpPosition);
        armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        armMotor.setPower(0.5);
    }

    // If the B button is pressed, lower the arm
    if (gamepad1.b) {
        armMotor.setTargetPosition(armDownPosition);
        armMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        armMotor.setPower(0.5);
    }

    // Get the current position of the armMotor
    double position = armMotor.getCurrentPosition();

    // Get the target position of the armMotor
    double desiredPosition = armMotor.getTargetPosition();

    // Show the position of the armMotor on telemetry
    telemetry.addData("Encoder Position", position);

    // Show the target position of the armMotor on telemetry
    telemetry.addData("Desired Position", desiredPosition);

    telemetry.update();
}
}
}

```

Blocks

