Utilizando encoders

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

- Programando encoders

Programando encoders

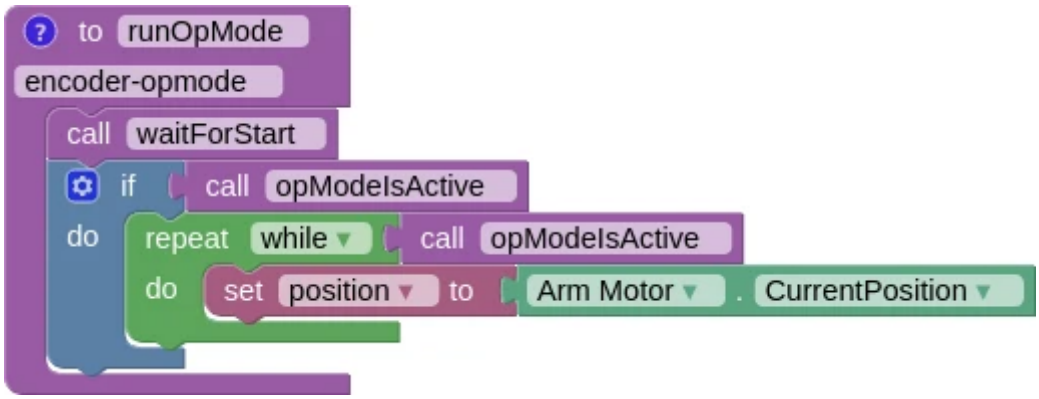
Leitura de encoders

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

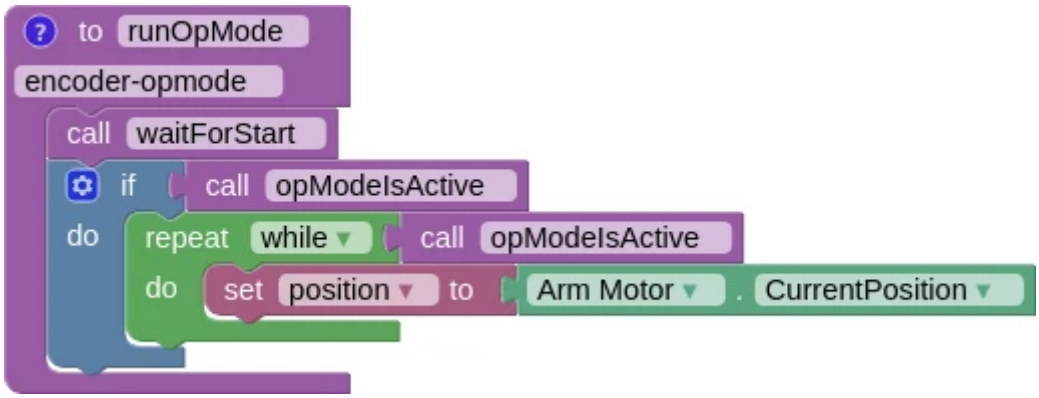


Embora seja conveniente usar o encoder de motor embutido, isso pode se tornar confuso ao usar encoders externos. Para usar encoders externos, você deve usar o objeto do motor associado à porta. Por exemplo, se houver um motor na porta 1 chamado "Arm Motor" e um encoder externo conectado à porta de encoder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

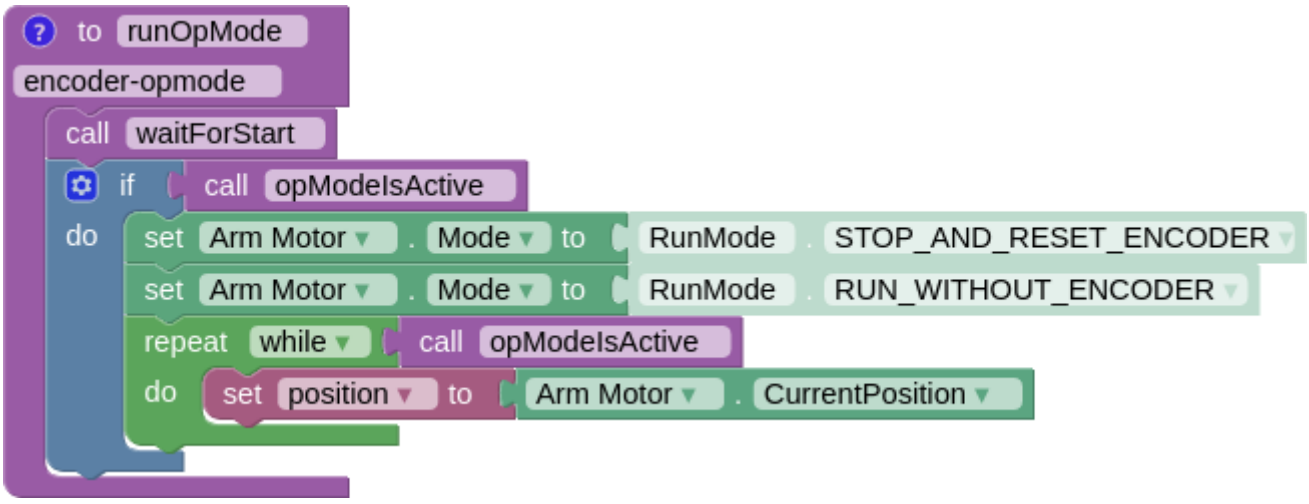


Ótimo! Agora temos a posição relativa do nosso encoder, reportada pelo número de "contagens" que ele se afastou do que considera ser o zero. No entanto, muitas vezes é conveniente fazer o encoder começar em zero no início do OpMode. Embora tecnicamente isso não mude nada, pode ajudar na depuração e simplificar 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);

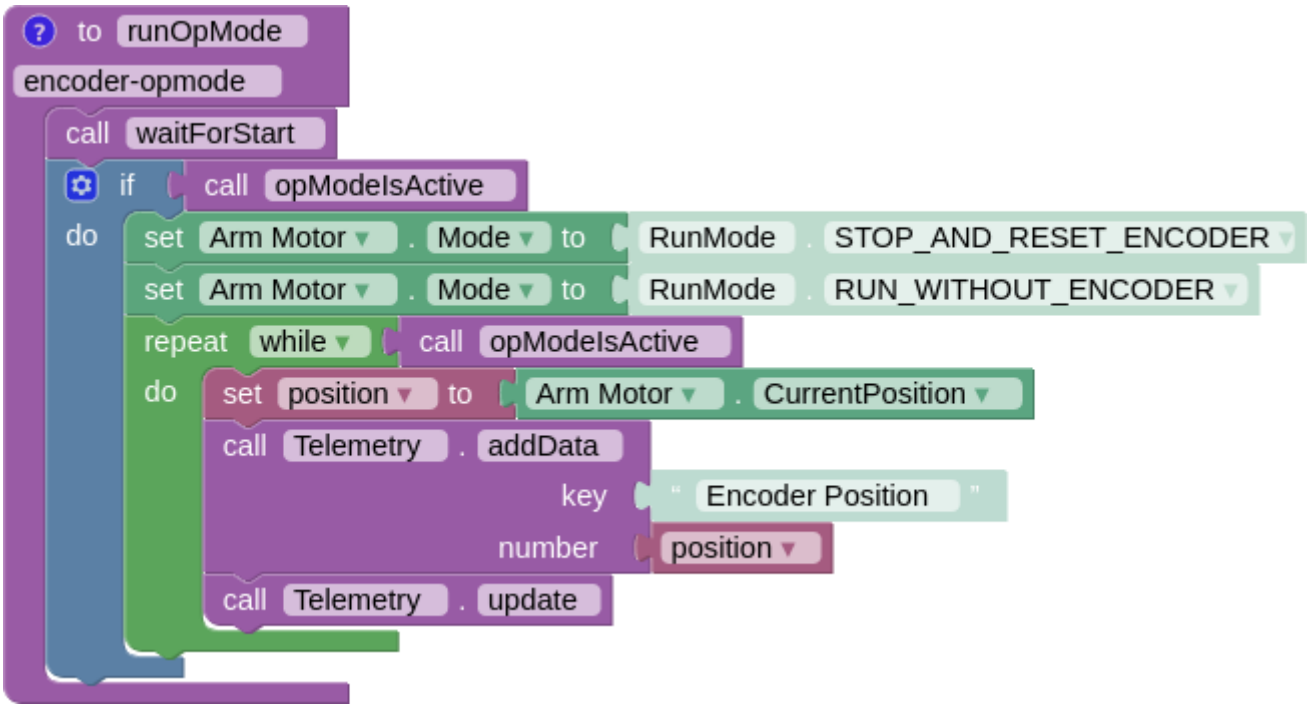
        // Turn the motor back on, required if you use STOP_AND_RESET_ENCODER
        motor.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);

        waitForStart();

        while (opModeIsActive()) {
            // 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.

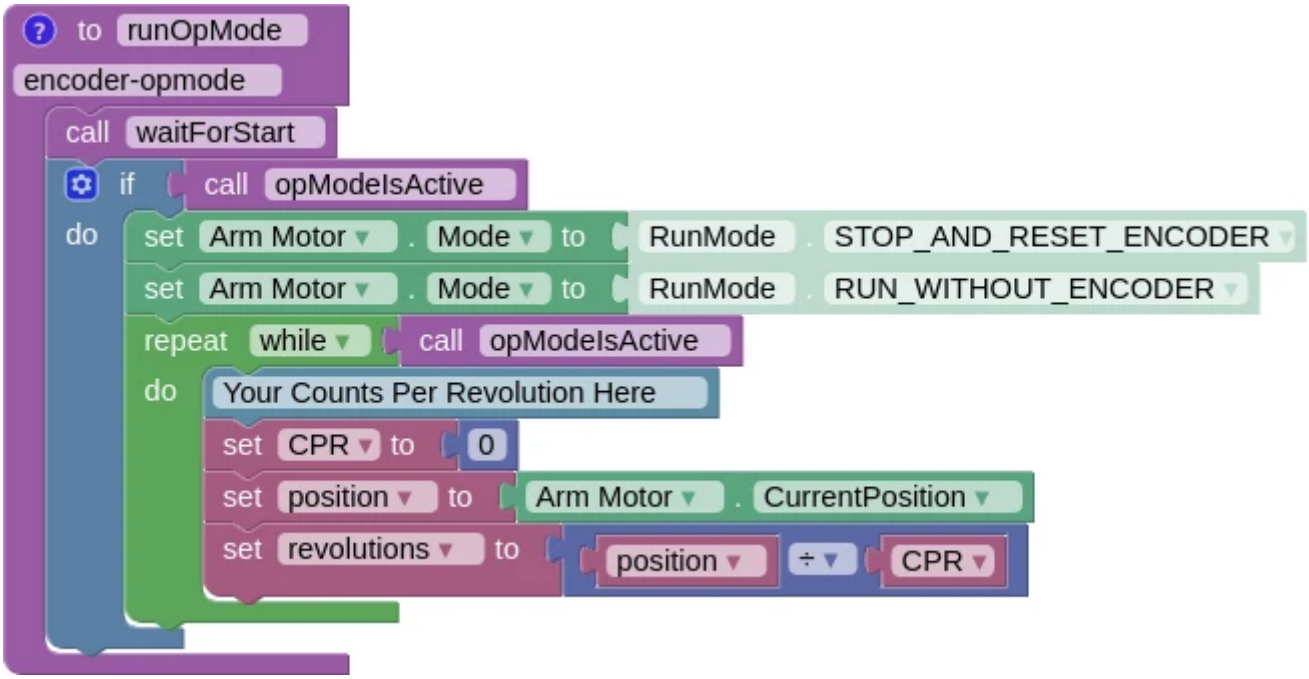
No exemplo a seguir, dividimos a posição do codificador pelo número de contagens por revolução para obter o número de revoluções que o codificador realizou. Você deve substituir [Your Counts Per Revolution Here] pelo número de contagens por revolução do seu motor, que pode ser encontrado na página do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

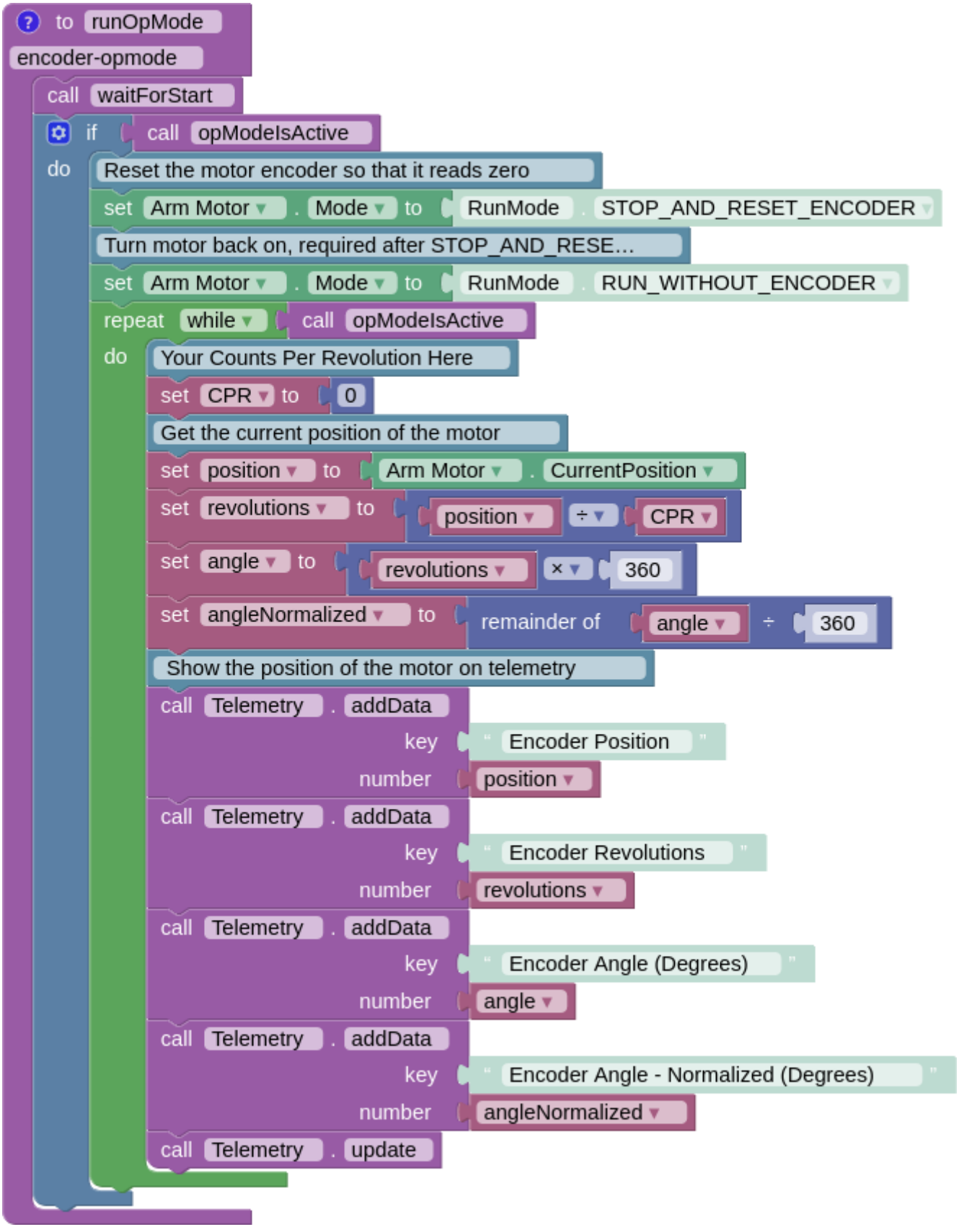
        while (opModeIsActive()) {
            double CPR = [Your Counts Per Revolution Here];

            // 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.

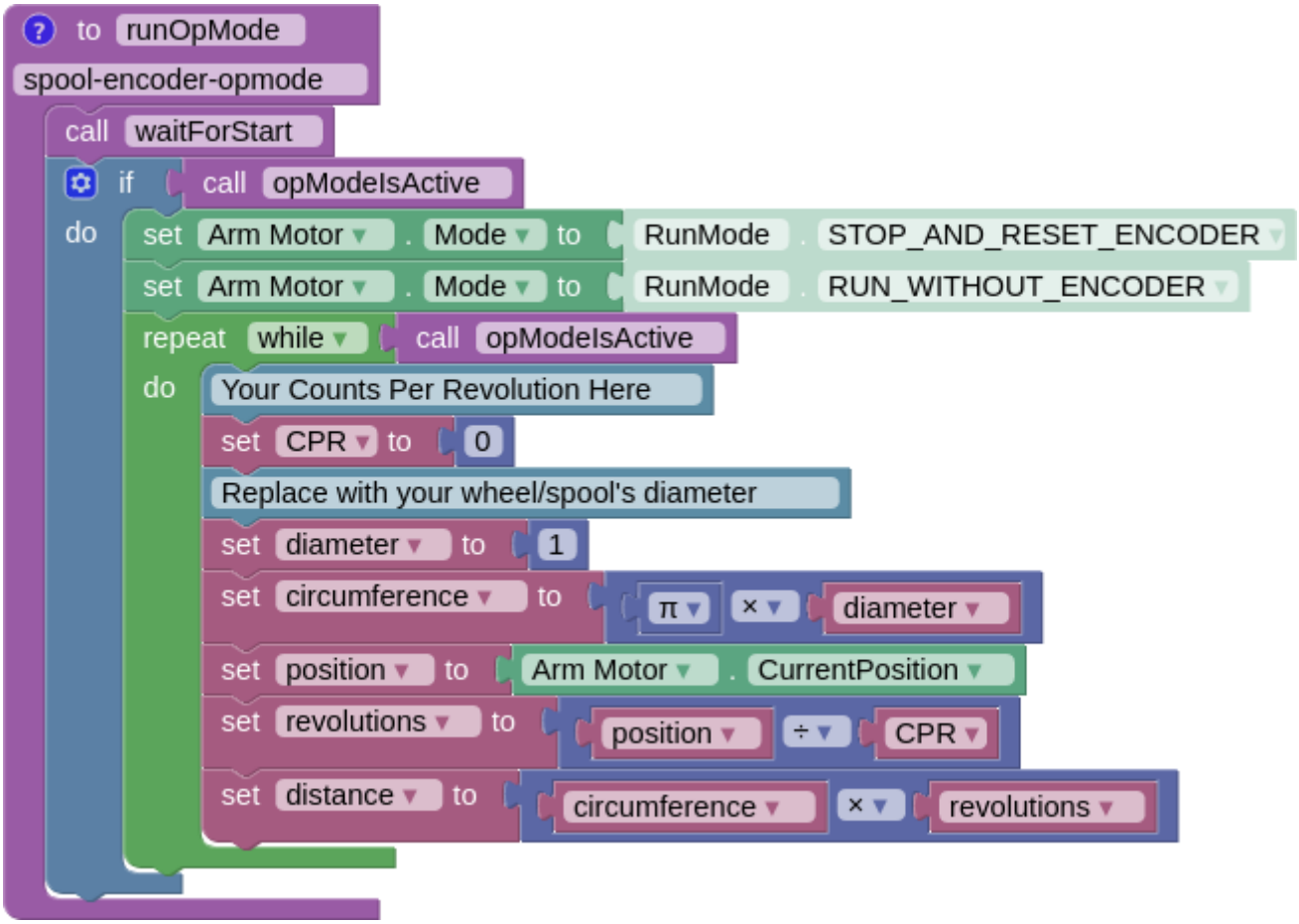
A partir daí, podemos calcular a circunferência (o comprimento do arco do círculo, ou a distância que a roda/carretel percorrerá em uma rotação).

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;

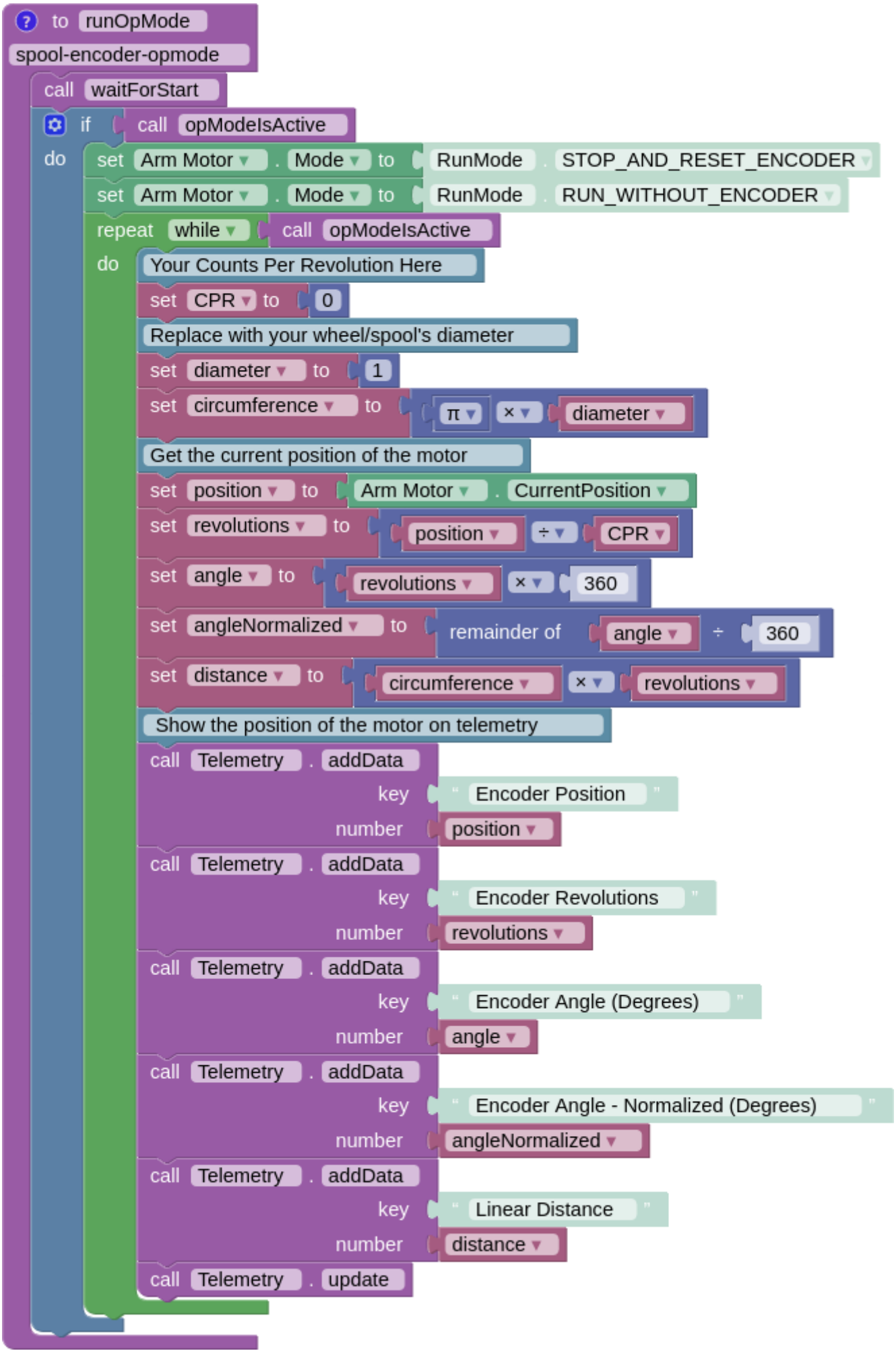
        // Get the current position of the motor
        int position = motor.getCurrentPosition();
        double revolutions = position/CPR;

        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


to runOpMode
spool-encoder-opmode
call waitForStart
if call opModeIsActive
do
set Arm Motor . Mode to RunMode . STOP_AND_RESET_ENCODER
set Arm Motor . Mode to RunMode . RUN_WITHOUT_ENCODER
repeat while call opModeIsActive
do
Your Counts Per Revolution Here
set CPR to 0
Replace with your wheel/spool's diameter
set diameter to 1
set circumference to π × diameter
Get the current position of the motor
set position to Arm Motor . CurrentPosition
set revolutions to position ÷ CPR
set angle to revolutions × 360
set angleNormalized to remainder of angle ÷ 360
set distance to circumference × revolutions
Show the position of the motor on telemetry
call Telemetry . addData
key " Encoder Position "
number position
call Telemetry . addData
key " Encoder Revolutions "
number revolutions
call Telemetry . addData
key " Encoder Angle (Degrees) "
number angle
call Telemetry . addData
key " Encoder Angle - Normalized (Degrees) "
number angleNormalized
call Telemetry . addData
key " Linear Distance "
number distance
call Telemetry . update

Operando motores com encoders

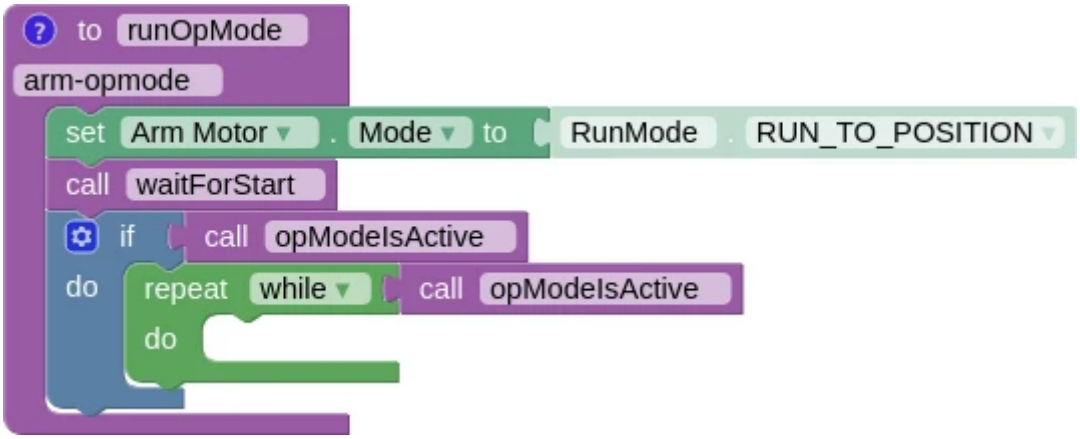
Aprendemos como ler os valores do codificador, mas como definimos onde queremos ir e dizemos ao motor para ir até lá?

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.

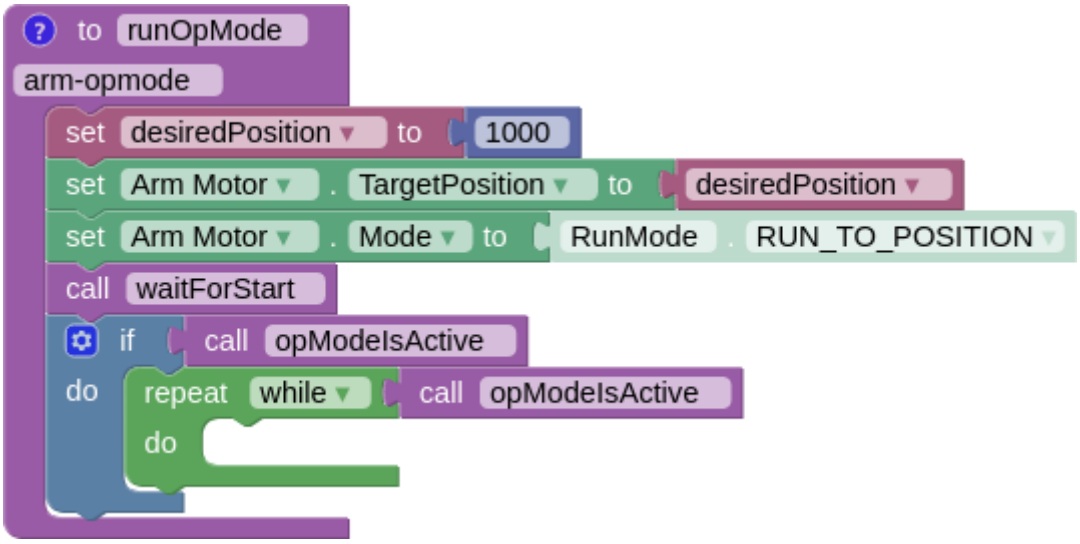
No entanto, antes de dizer ao motor para ir até uma posição, precisamos informar ao motor qual posição ele deve atingir. Observe que esse valor deve ser um número inteiro. Vamos modificar o código acima para fazer isso.

> 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



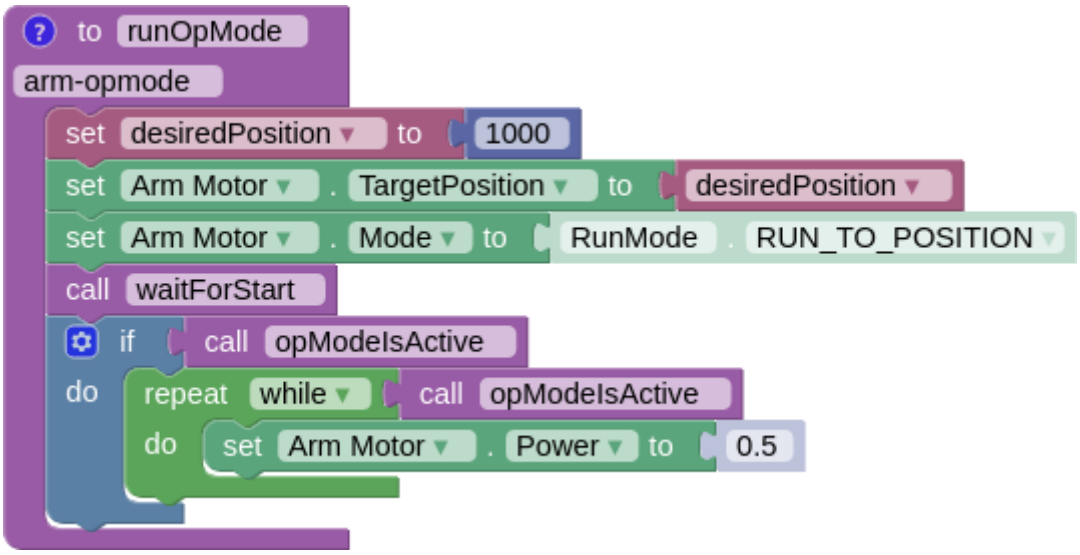
Este código diz ao motor para se mover até 1000 contagens, usando um loop PID para controlar a posição do motor. Você pode ler mais sobre loops PID aqui.

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 (opModeIsActive()) {
            // 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();

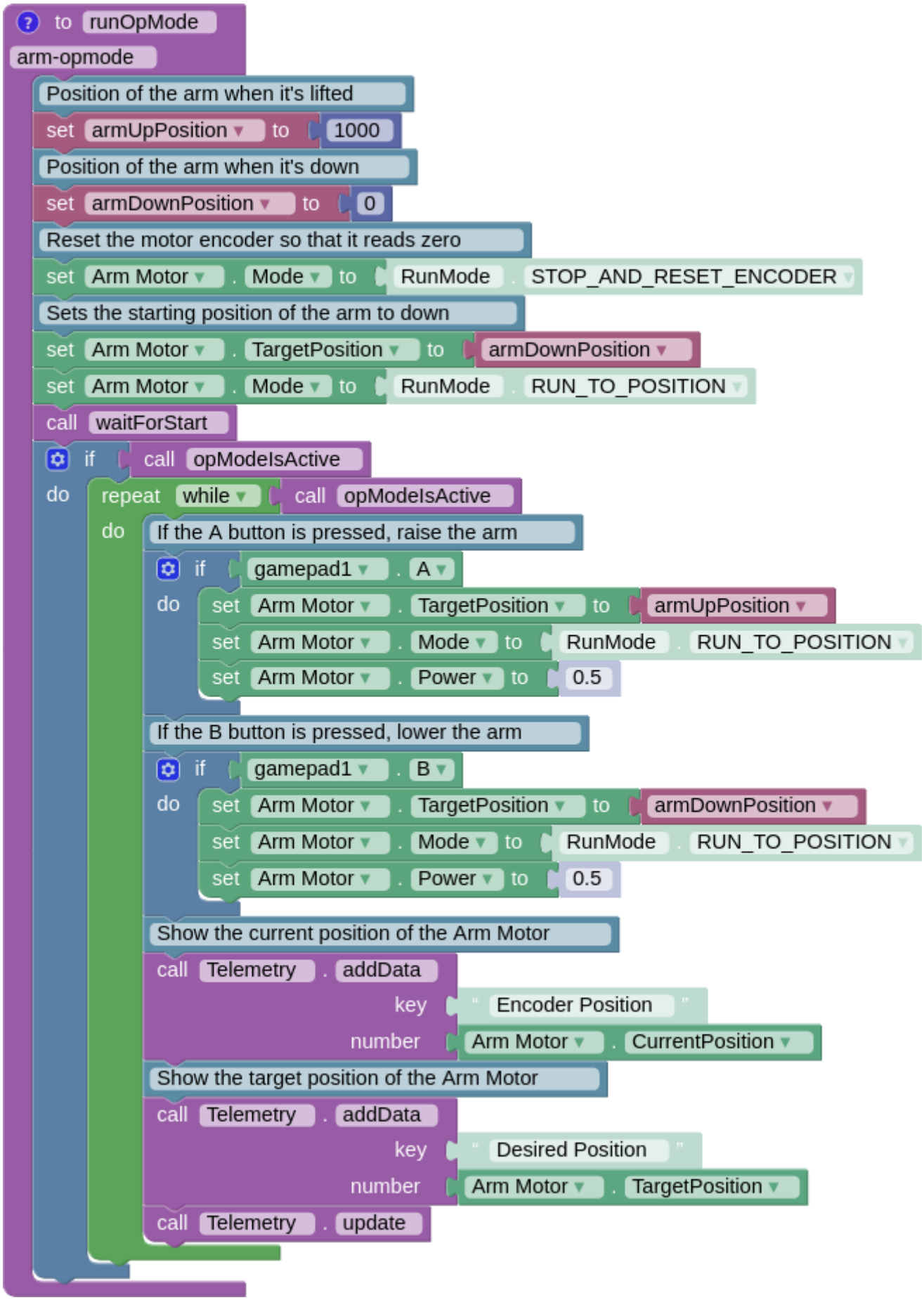
            // Get the target position of the armMotor
            double desiredPosition = armMotor.getTargetPosition();

            // 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


to runOpMode
arm-opmode
Position of the arm when it's lifted
set armUpPosition to 1000
Position of the arm when it's down
set armDownPosition to 0
Reset the motor encoder so that it reads zero
set Arm Motor . Mode to RunMode . STOP_AND_RESET_ENCODER
Sets the starting position of the arm to down
set Arm Motor . TargetPosition to armDownPosition
set Arm Motor . Mode to RunMode . RUN_TO_POSITION
call waitForStart
if call opModeIsActive
do
repeat while call opModeIsActive
do
If the A button is pressed, raise the arm
if gamepad1 . A
do
set Arm Motor . TargetPosition to armUpPosition
set Arm Motor . Mode to RunMode . RUN_TO_POSITION
set Arm Motor . Power to 0.5
If the B button is pressed, lower the arm
if gamepad1 . B
do
set Arm Motor . TargetPosition to armDownPosition
set Arm Motor . Mode to RunMode . RUN_TO_POSITION
set Arm Motor . Power to 0.5
Show the current position of the Arm Motor
call Telemetry . addData
key " Encoder Position "
number Arm Motor . CurrentPosition
Show the target position of the Arm Motor
call Telemetry . addData
key " Desired Position "
number Arm Motor . TargetPosition
call Telemetry . update