# OnBot Java

O OnBot Java é uma ferramenta de programação baseada em texto que permite aos programadores usar um navegador da web para criar, editar e salvar seus modos de operação Java (op modes). Nesta seção, os usuários podem aprender como criar um op mode, bem como os conceitos básicos de programação dos atuadores e sensores apresentados no banco de testes.

Siga o guia para obter uma compreensão aprofundada de como trabalhar com o OnBot Java ou navegue até a seção que atenda às suas necessidades:

| Seção | Objetivos da seção |
|---|---|
| Criando um Op Mode | Concentre-se em como navegar na interface do OnBot Java e criar um op mode. |
| Fundamentos da programação | Explora a estrutura e os elementos-chave necessários para um modo de operação (op mode), assim como alguns dos componentes essenciais do Java. |
| Programando acionadores | Como codificar servos e motores. Esta seção guia através da lógica básica de codificar atuadores, controlar atuadores com um gamepad e utilizar telemetria. |
| Programando sensores | Como codificar um dispositivo digital. Esta seção concentra-se na lógica básica de programar um dispositivo digital, como um Sensor de Toque REV. |

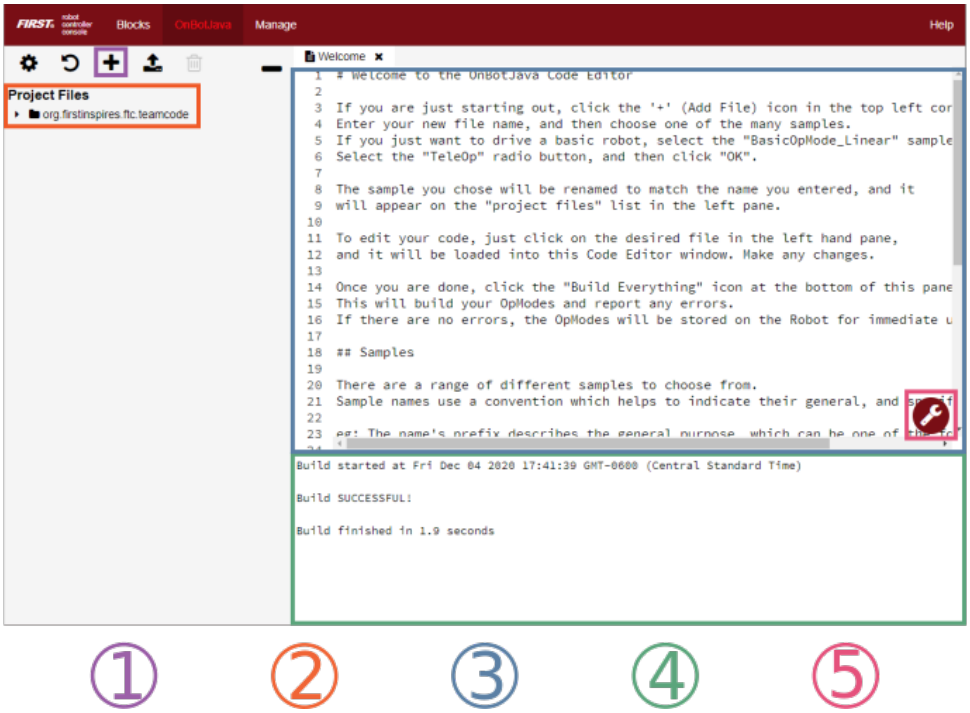
# Criando um Op Mode

Antes de mergulhar e criar o seu primeiro modo de operação (op mode), você deve considerar o conceito de convenções de nomenclatura. Ao escrever código, o objetivo é ser o mais claro possível sobre o que está acontecendo dentro do código. É aqui que entra o conceito de convenções de nomenclatura. Convenções de nomenclatura comuns foram estabelecidas pelo mundo da programação para denotar variáveis, classes, funções, etc. Os modos de operação compartilham algumas semelhanças com as classes. Assim, a convenção de nomenclatura para modos de operação tende a seguir a convenção de nomenclatura para classes; onde a primeira letra de cada palavra é maiúscula.

> Esta seção pressupõe que você já acessou a plataforma OnBot Java durante a introdução ao "Olá Robô - Programação". Se você não souber como acessar o OnBot Java, por favor, reveja

esta seção antes de continuar.

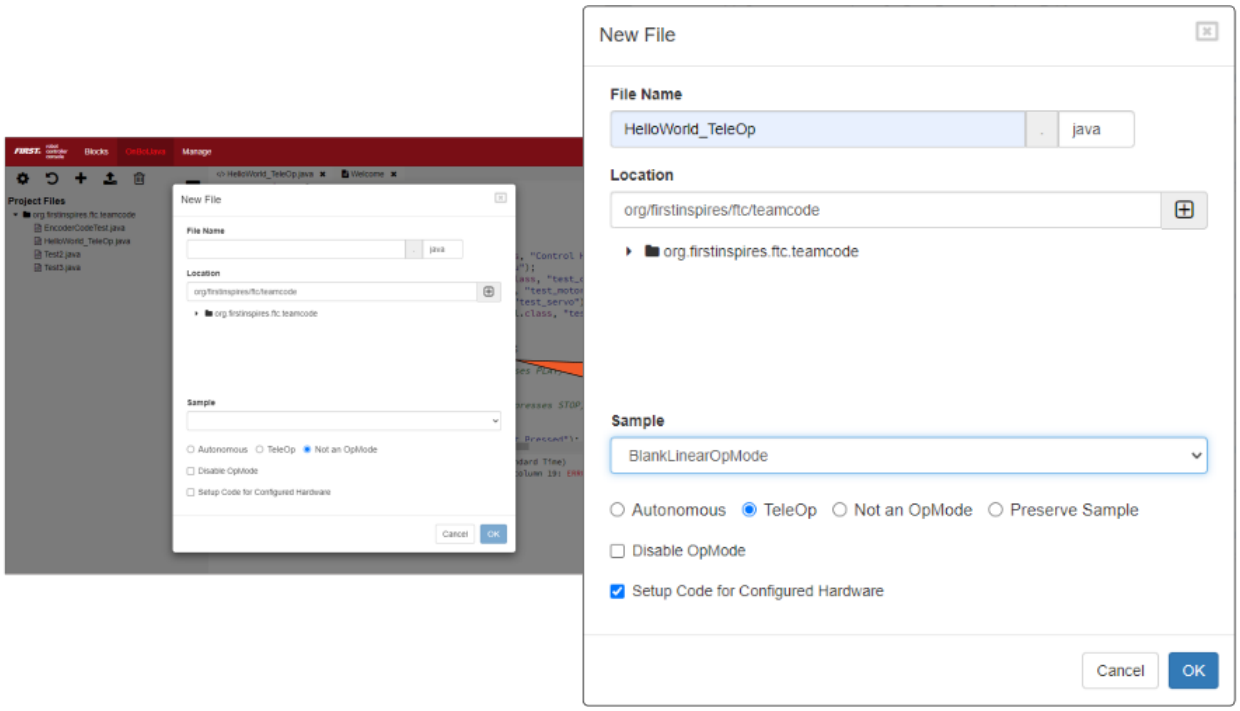
Para começar, acesse o Robot Controller Console e vá para a página do OnBot Java. Há algumas coisas importantes para observar na página principal do OnBot Java.




1. Criar Novo Modo de Operação - O botão de sinal de adição abre uma janela para criar um novo modo de operação.
2. Painel do Navegador de Projetos - Este painel mostra todos os arquivos de projeto Java no Controlador do Robô.
3. Painel de Edição de Código-Fonte - Esta área é a principal para edição de código.
4. Painel de Mensagens - Este painel fornece mensagens sobre o sucesso ou falha na compilação do código.
5. Compilar Tudo - Compila todos os arquivos .java no Controlador do Robô.

Quando um modo de operação é criado ou editado, o editor OnBot Java salvará automaticamente o arquivo .java no sistema de arquivos do Controlador do Robô. No entanto, para executar o código no Controlador do Robô, o arquivo de texto .java precisa ser convertido para um binário que pode ser carregado dinamicamente no aplicativo FTC Robot Controller. Essa conversão é feita compilando os modos de operação.

Selecione o botão Create new Op Mode. Isso abrirá a janela New File. Esta janela permite que os usuários escolham configurações como: nomear seus modos op, selecionar um código de exemplo para desenvolver, ou escolher o tipo de modo op. Para este guia, selecione as seguintes seções:




- File Name: HelloRobot_TeleOp
- Sample: BlankLinearOpMode
- Op Mode Type: TeleOp
- Setup for Configured Hardware: on

Após as configurações adequadas terem sido escolhidas, selecione "OK" para criar o modo op. O novo arquivo será exibido no Painel de Navegação do Projeto.

# Fundamentos da programação

Durante o processo de criação de um modo op, a ferramenta OnBot Java oferecia várias opções para escolher. Essas opções definem quais informações já estão incluídas no modo op, o que pode simplificar o que um programador precisa fazer do seu lado. Por exemplo, uma opção foi dada para selecionar um exemplo. No OnBot Java, esses exemplos atuam como modelos; fornecendo declarações, estrutura lógica e sintaxe para diferentes casos de uso em robótica.

Na seção anterior, as seguintes configurações foram selecionadas: a opção Setup Code for Configured Hardware, a opção TeleOp e um exemplo de código chamado BlankLinearOpMode. Essas opções combinadas configuram a estrutura básica de código necessária para ter um modo op funcional.

Um modo op é considerado um conjunto de instruções para um robô seguir a fim de entender o mundo ao seu redor. Embora o SDK forneça estruturas prontas para modos op, entender quais conceitos o modelo está utilizando e por que, ajuda a aumentar o conhecimento em programação. Siga nesta seção para aprender mais sobre o modelo de modo op e os conceitos de programação que compõem sua estrutura.

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

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.Blinker;
import com.qualcomm.robotcore.hardware.Gyroscope;
import com.qualcomm.robotcore.hardware.ColorSensor;
import com.qualcomm.robotcore.hardware.Servo;
import com.qualcomm.robotcore.hardware.DigitalChannel;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.eventloop.opmode.Disabled;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorSimple;
import com.qualcomm.robotcore.util.ElapsedTime;

@TeleOp

public class HelloWorld_TeleOp extends LinearOpMode {
    private Gyroscope imu;
    private ColorSensor test_color;
    private DcMotor test_motor;
    private Servo test_servo;
    private DigitalChannel test_touch;


    @Override
    public void runOpMode() {
        imu = hardwareMap.get(Gyroscope.class, "imu");
        test_color = hardwareMap.get(ColorSensor.class, "test_color");
        test_motor = hardwareMap.get(DcMotor.class, "test_motor");
        test_servo = hardwareMap.get(Servo.class, "test_servo");
        test_touch = hardwareMap.get(DigitalChannel.class, "test_touch");

        telemetry.addData("Status", "Initialized");
        telemetry.update();
        // Wait for the game to start (driver presses PLAY)
        waitForStart();

        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {
            telemetry.addData("Status", "Running");
```

```
        telemetry.update();


    }
}
```

> O bloco de código fornece a estrutura do modelo de modo op com base na Configuração Hello Robot e com alguns comentários ausentes. Se outra configuração estiver sendo usada, o código será ligeiramente diferente, mas muitos dos conceitos subjacentes são os mesmos.

# Conceitos de programação

No início do modo op, há uma anotação que ocorre antes da definição da classe. Essa anotação afirma que este é um modo op teleoperado (ou seja, controlado pelo motorista):]

```
@TeleOp
```

Em Java, as anotações são metadados ou informações descritivas sobre o código. Neste caso, a anotação está sendo usada para informar ao sistema que este modo de operação é tele-operado. Alterar a anotação de @TeleOp para @Autonomous modificará o código para um modo de operação autônomo.

```
public class HelloWorld_TeleOp extends LinearOpMode {
```

Você também pode observar que o editor OnBot Java criou cinco variáveis de membro privado para este modo de operação. Essas variáveis irão conter referências aos cinco dispositivos configurados que o editor OnBot Java detectou na configuração ativa.

```
private Gyroscope imu;
private ColorSensor test_color;
private DcMotor test_motor;
private Servo test_servo;
private DigitalChannel test_touch;
```

Em seguida, há um método sobrescrito chamado runOpMode. Todo modo de operação do tipo LinearOpMode deve implementar este método. Este método é chamado quando um usuário seleciona e executa o modo de operação.

```
    @Override
    public void runOpMode() {
```

O mapeamento de hardware foi introduzido na seção de configuração, como um processo de duas partes. A primeira parte do processo consistiu em criar um arquivo de configuração. A segunda parte do processo é obter referências aos dispositivos de hardware a partir do objeto hardwareMap.

> O objeto hardwareMap está disponível para uso no método runOpMode. Trata-se de um objeto do tipo classe hardwareMap.

No início do método runOpMode, o modo de operação utiliza o objeto hardwareMap para obter referências aos dispositivos de hardware listados no arquivo de configuração do Controlador do Robô:

```
imu = hardwareMap.get(Gyroscope.class, "imu");
test_color = hardwareMap.get(ColorSensor.class, "test_color");
test_motor = hardwareMap.get(DcMotor.class, "test_motor");
test_servo = hardwareMap.get(Servo.class, "test_servo");
test_touch = hardwareMap.get(DigitalChannel.class, "test_touch");
```

A chamada do método hardwareMap.get() é utilizada para recuperar os dispositivos de hardware e atribuí-los a variáveis. A chamada do método aceita dois argumentos: uma referência à classe específica de dispositivos de hardware à qual o dispositivo pertence e o nome do dispositivo de hardware no arquivo de configuração. O nome fornecido no hardwareMap.get() precisa corresponder ao nome do dispositivo no arquivo de configuração. Se os nomes não coincidirem, o modo de operação gerará um erro em tempo de execução indicando que não é possível encontrar o dispositivo.

> Para mais infomrações sobre erros de execução cheque nossa seção de Erros comuns

Nas próximas declarações do exemplo, o modo de operação solicita ao usuário que pressione o botão de início para continuar. Ele utiliza outro objeto disponível no método runOpMode. Esse objeto é chamado de "telemetry", e o modo de operação utiliza o método addData para adicionar uma mensagem a ser enviada para a Estação do Motorista. Em seguida, o modo de operação chama o método update para enviar a mensagem para a Estação do Motorista. Depois disso, ele chama o método waitForStart para aguardar até que o usuário pressione o botão de início na estação do motorista para iniciar a execução do modo de operação.

> Telemetria é o processo de coleta e transmissão de dados. Na robótica, a telemetria é frequentemente utilizada para enviar dados internos provenientes de atuadores e sensores para a Estação do Motorista. Esses dados podem ser analisados pelos usuários para tomar decisões que possam aprimorar o código.

```
telemetry.addData("Status", "Initialized");
telemetry.update();
// Wait for the game to start (driver presses PLAY)
waitForStart();
```

> Todas as operações lineares (linear op modes) devem conter uma instrução waitForStart para garantir que o robô não começará a executar o modo de operação até que o motorista pressione o botão de início.

Após receber um comando de início, o modo de operação entra em um loop while e continua iterando neste loop até que o modo de operação não esteja mais ativo (ou seja, até que o usuário pressione o botão de parada na Estação do Motorista):

```
// run until the end of the match (driver presses STOP)
while (opModeIsActive()) {
    telemetry.addData("Status", "Running");
    telemetry.update();


}
```

Conforme o modo de operação itera no loop while, ele continuará a enviar mensagens de telemetria com o índice "Status" e a mensagem "Running" para serem exibidas na Estação do Motorista.

# Sintaxe

Linguagens de programação, assim como qualquer idioma, possuem um conjunto de regras e princípios orientadores que permitem que as declarações sejam compreendidas universalmente. Elementos como pontuação, estrutura de palavras e formatação desempenham um papel na interpretação de uma linha de código. Em linguística e ciência da computação, as regras que governam a estrutura de uma sentença são conhecidas como sintaxe.

É importante compreender a sintaxe do Java, pois erros de sintaxe serão comuns e difíceis de rastrear sem um nível básico de entendimento.

# Programação Orientada a objetos

Esta seção fez várias referências a métodos, objetos e classes. Todos esses são tópicos de programação intermediários a avançados, frequentemente centrados no conceito de programação orientada a objetos. O objetivo do guia Olá Robô é servir como um curso introdutório para programação de robôs, em vez de aprofundar-se em conceitos de programação.

No entanto, mantenha a programação orientada a objetos em mente à medida que suas habilidades se desenvolvem. Por enquanto, a coisa mais importante a saber é que, ocasionalmente, métodos dentro das bibliotecas do SDK precisarão ser chamados para realizar uma determinada tarefa. Por exemplo, a linha OláRobo_TeleOp.opModeIsActive() chama o método opModeIsActive, que é o procedimento no SDK capaz de indicar quando o modo de operação foi ativado pelo telefone da estação do motorista.

À medida que avançarmos, grande parte do código específico para motores, servos ou sensores lidará com chamadas a outros métodos ou classes.

> For more information on classes and methods in the SDK check out the Java Doc.

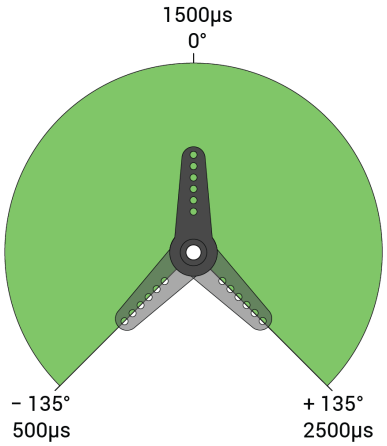
# Programando acionadores

## Noções básicas do servo

O objetivo desta seção é abordar alguns conceitos básicos de programação de um servo no OnBot Java. Ao final desta seção, os usuários devem ser capazes de controlar um servo com um gamepad, bem como entender algumas das necessidades fundamentais de programação relacionadas ao servo.

> Esta seção está considerando o Servo Robô Inteligente no seu modo padrão. Se o seu servo foi alterado para funcionar em modo contínuo ou com limites angulares, ele não se comportará da mesma forma usando os exemplos de código abaixo. Você pode aprender mais sobre o Servo Robô Inteligente ou alterar o modo do servo através do SRS Programmer clicando nos hiperlinks.

Com um servo típico, você pode especificar uma posição alvo para o servo. O servo girará seu eixo do motor para mover-se até a posição alvo e, em seguida, manterá essa posição, mesmo que forças moderadas sejam aplicadas para tentar perturbar sua posição.




Tanto para o Blocks quanto para o OnBot Java, você pode especificar uma posição alvo que varia de 0 a 1 para um servo. Para um servo com uma faixa de 270°, se a faixa de entrada fosse de 0 a 1, então um sinal de entrada de 0 faria com que o servo se movesse para -135°. Para um sinal de entrada de 1, o servo se moveria para +135°. Entradas entre o mínimo e o máximo têm ângulos correspondentes distribuídos uniformemente entre o ângulo mínimo e máximo do servo. Isso é importante ter em mente enquanto aprende a programar servos.

# Programando um Servo

Adicione a seguinte linha ao loop while do Op Mode: test_servo.setPosition(1);

Como segue abaixo:

```
while (opModeIsActive()) {
    test_servo.setPosition(1);
    telemetry.addData("Status", "Running");
    telemetry.update();


}
```

Selecione *Build everything* para compilar o código

> Execute este modo operacional (op mode) no banco de testes duas vezes e considere as seguintes perguntas:
>
> - O servo motor se moveu durante a primeira execução?
> - O servo motor se moveu durante a segunda execução?
> - Se o servo motor não se moveu, altere test_servo.setPosition(1); para test_servo.setPosition(0); e tente novamente.

A intenção do test_servo.setPosition(); é definir a posição do servo. Se o servo já estiver na posição definida quando o código é executado, ele não mudará de posição. Vamos tentar adicionar a linha test_servo.setPosition(0); ao código na seção de inicialização.

```
public void runOpMode() {
        imu = hardwareMap.get(Gyroscope.class, "imu");
        test_color = hardwareMap.get(ColorSensor.class, "test_color");
        test_motor = hardwareMap.get(DcMotor.class, "test_motor");
        test_servo = hardwareMap.get(Servo.class, "test_servo");
        test_touch = hardwareMap.get(DigitalChannel.class, "test_touch");

        test_servo.setPosition(0);

        telemetry.addData("Status", "Initialized");
        telemetry.update();
        // Wait for the game to start (driver presses PLAY)
        waitForStart();

        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {
            test_servo.setPosition(1);
            telemetry.addData("Status", "Running");
            telemetry.update();

        }
    }
}
```

> Tente executar este modo operacional no banco de testes. Dê algum tempo entre pressionar "init" e pressionar "play" e considere a seguinte pergunta: O que é diferente da execução anterior?

A linha test_servo.setPosition(0); que foi adicionada na etapa anterior altera a posição do servo para 0 durante a fase de inicialização, então, quando o modo operacional é executado, o servo sempre se moverá para a posição 1. Para algumas aplicações, iniciar o servo em um estado conhecido, como na posição zero, é benéfico para o funcionamento de um mecanismo. Definir o servo no estado conhecido durante a inicialização garante que ele esteja na posição correta quando o modo operacional é executado.

# Programando um servo com um controle

O foco deste exemplo é atribuir determinadas posições de servo a botões no gamepad. Para este exemplo, o estado conhecido permanecerá na posição 0, de modo que, após a inicialização, o servo estará na posição de -135 graus no intervalo do servo. A lista a seguir mostra quais botões correspondem a quais posições do servo.

| Botões | Posição em graus | Posiçaõ no código |
|---|---|---|
| Y | -135 | 0 |
| X | 0 | 0.5 |
| B | 0 | 0.5 |
| A | 135 | 1 |

A melhor maneira de alternar a posição do servo será usar uma instrução condicional if/else if. Uma instrução if avalia se uma afirmação condicional é verdadeira ou falsa. Se a afirmação condicional for verdadeira, uma ação definida (como o movimento do servo) é realizada. Se a afirmação condicional for falsa, a ação não é realizada.

Uma instrução if/else if aceita várias afirmações condicionais diferentes. Se a primeira afirmação condicional for falsa, então a segunda afirmação condicional é analisada. Para entender melhor esse conceito, considere o seguinte código:

```
if (gamepad1.y){
    //move to -135 degrees
    test_servo.setPosition(0);

} else if (gamepad1.x || gamepad1.b) {
    //move to 0 degrees
    test_servo.setPosition(0.5);

} else if (gamepad1.a) {
    //move to 135 degrees
    test_servo.setPosition(1);

}
```

Existem três caminhos diferentes nesta instrução if/else if. Se a primeira afirmação condicional for verdadeira (o botão Y está pressionado), o servo se move para a posição 0 e as outras afirmações condicionais são ignoradas. Se a primeira condição for falsa (o botão Y não está pressionado), a

segunda condição é analisada. Esse comportamento se repete até que uma condição seja atendida ou todas as condições tenham sido testadas e consideradas falsas.

> || é um operador lógico em Java. Esse símbolo é o equivalente ao "ou". Usando isso como um parâmetro condicional, ele vai verificar se o botão x ou b estão pressionados para ser verdadeira.

```
public void runOpMode() {
        imu = hardwareMap.get(Gyroscope.class, "imu");
        test_color = hardwareMap.get(ColorSensor.class, "test_color");
        test_motor = hardwareMap.get(DcMotor.class, "test_motor");
        test_servo = hardwareMap.get(Servo.class, "test_servo");
        test_touch = hardwareMap.get(DigitalChannel.class, "test_touch");

        test_servo.setPosition(0);

        telemetry.addData("Status", "Initialized");
        telemetry.update();
        // Wait for the game to start (driver presses PLAY)
        waitForStart();

        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {
           if (gamepad1.y){
              //move to -135 degrees
              test_servo.setPosition(0);

           } else if (gamepad1.x || gamepad1.b) {
              //move to 0 degrees
              test_servo.setPosition(0.5);

           } else if (gamepad1.a) {
              //move to 135 degrees
              test_servo.setPosition(1);
                    }

           telemetry.addData("Status", "Running");
           telemetry.update();

        }
```

```
    }
}
```

# Servos e telemetria

Lembrando que telemetria é o processo de coletar e transmitir dados. Na robótica, a telemetria é usada para enviar dados internos de atuadores e sensores para a Estação do Motorista (Driver Station). Esses dados podem ser analisados pelos usuários para tomar decisões que podem aprimorar o código.

A telemetria mais útil do servo é a posição do servo ao longo de sua faixa de 270 graus. Para obter essa informação, a seguinte linha precisa ser usada.

```
test_servo.getPosition();
```

Na seção de fundamentos de programação, a linha telemetry.addData(); foi discutida brevemente. Essa chamada de método recebe um parâmetro de chave (key) e variável e envia as informações para a Estação do Motorista. A chave é uma string, ou uma linha de texto, que deve definir a variável. Neste caso, telemetry.addData(); está sendo usado para enviar para a Estação do Motorista a posição do servo conforme ela é alterada, então a chave pode ser "Posição do Servo". O parâmetro, no entanto, será a chamada do método test_servo.getPosition();.

```
double motorPower = 0;
while (opModeIsActive()) {
        if (gamepad1.y){
            //move to -135 degrees
            test_servo.setPosition(0);

         } else if (gamepad1.x || gamepad1.b) {
            //move to 0 degrees
            test_servo.setPosition(0.5);

         } else if (gamepad1.a) {
            //move to 135 degrees
            test_servo.setPosition(1);

         telemetry.addData("Servo Position", test_servo.getPosition());
         telemetry.addData("Status", "Running");
         telemetry.update();

      }
```

# Noções básicas de motor

> Modifique o seu modo de operação (op mode) para adicionar o código relacionado ao motor. Isso pode ser feito limpando as modificações atuais do seu código ou adicionando o código relacionado ao motor ao seu modo de operação atual.

O objetivo desta seção é abordar alguns conceitos básicos de programação de um motor no ambiente OnBot Java. Ao final desta seção, os usuários deverão ser capazes de controlar um motor usando um gamepad, bem como compreender alguns dos fundamentos ao lidar com codificadores de motor.

## Controlando motores

Adicione a linha test_motor.setPower(1); ao loop while do modo de operação (op mode).

```
while (opModeIsActive()) {
    test_motor.setPower(1);


    telemetry.addData("Status", "Running");
    telemetry.update();


}
```

Selecione *Build Everything* para compilar o código

> Tente executar este modo de operação no banco de testes e considere as seguintes perguntas:
>
> - A que velocidade o motor está funcionando?
> - O que acontece se você alterar a potência de 1 para 0.3?
> - O que acontece se você alterar a potência para -1?

O nível de potência enviado ao motor depende do número numérico atribuído ao motor. A mudança de 1 para 0,3 diminuiu a velocidade do motor de 100% do ciclo de trabalho para 30% do ciclo de trabalho. Enquanto isso, a mudança para -1 permitiu que o motor girasse a 100% do ciclo de trabalho na direção oposta. Portanto, a potência pode ser variada para mover um motor para frente ou para trás. No entanto, a linha test_motor.setPower(1); fará com que o motor funcione na direção atribuída até que algo no código pare o motor ou cause uma mudança na direção.

## Controlando motores com um gamepad

Na seção anterior, você aprendeu como configurar o motor para funcionar em um nível de potência específico em uma direção específica. No entanto, em algumas aplicações, pode ser necessário controlar o motor com um gamepad para alterar facilmente a direção ou o nível de potência de um mecanismo.

Para esta seção, vamos criar uma variável dupla chamada motorPower. Essa variável será criada dentro do modo de operação (op mode), mas fora do loop while.

```
public void runOpMode() {
        imu = hardwareMap.get(Gyroscope.class, "imu");
        test_color = hardwareMap.get(ColorSensor.class, "test_color");
        test_motor = hardwareMap.get(DcMotor.class, "test_motor");
        test_servo = hardwareMap.get(Servo.class, "test_servo");
        test_touch = hardwareMap.get(DigitalChannel.class, "test_touch");

        double motorPower = 0;

        telemetry.addData("Status", "Initialized");
        telemetry.update();
        // Wait for the game to start (driver presses PLAY)
        waitForStart();
        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {

            telemetry.addData("Status", "Running");
            telemetry.update();

        }
    }
}
```

Um tipo de dado double é um tipo numérico que pode armazenar números com pontos decimais. Dado que a potência, ou ciclo de trabalho, do motor opera em uma escala entre 1 e -1, a variável motorPower precisará ser capaz de armazenar dados numéricos com pontos decimais.

Considere as seguintes linhas de código:

```
  motorPower = - this.gamepad1.left_stick_y;
  test_motor.setPower(motorPower);
```

A linha motorPower = -this.gamepad1.left_stick_y; recebe uma entrada numérica que corresponde à posição do joystick do gamepad conforme ele se move ao longo do eixo y, e a atribui à variável

motorPower. A próxima linha, test_motor.setPower(motorPower);, define a potência do motor como sendo igual à variável motorPower.

> Observe que, para os gamepads Logitech F310, o valor Y de um joystick varia de -1, quando o joystick está em sua posição mais alta, a +1, quando o joystick está em sua posição mais baixa. Para alterar a relação direcional entre o motor e o joystick, de modo que a posição mais alta do joystick esteja correlacionada com a direção para frente do motor, é necessário usar um símbolo negativo ou o operador de negação.

```
// run until the end of the match (driver presses STOP)
double motorPower = 0;
while (opModeIsActive()) {
        motorPower = - this.gamepad1.left_stick_y;
        test_motor.setPower(motorPower);

        telemetry.addData("Status", "Running");
        telemetry.update();

    }
```

# Motores e telemetria

Lembre-se de que telemetria é o processo de coletar e transmitir dados. Na robótica, a telemetria é usada para enviar dados internos de atuadores e sensores para a Estação do Condutor (Driver Station). Esses dados podem então ser analisados pelos usuários para tomar decisões que podem melhorar o código.

Uma das formas mais comuns de dados de telemetria de motores é a informação retirada do encoder do motor. Motores DC REV, como o Core Hex Motor, são equipados com encoders internos que transmitem informações de posição na forma de contagens. Para obter informações dos encoders, a seguinte linha precisa ser utilizada:

```
test_motor.getCurrentPosition();
```

Na seção de fundamentos de programação, a linha telemetry.addData(); foi brevemente discutida. Essa chamada de método recebe um parâmetro de chave e variável e envia as informações para a Estação do Condutor (Driver Station). A chave é uma string, ou uma linha de texto, que deve definir a variável. Neste caso, o telemetry.addData(); está sendo usado para exibir a posição do motor na forma de contagens do encoder, então a chave pode ser "Encoder Value." No entanto, o parâmetro será a chamada de método test_motor.getCurrentPosition();.

```
double motorPower = 0;
while (opModeIsActive()) {
        motorPower = - this.gamepad1.left_stick_y;
        test_motor.setPower(motorPower);

        telemetry.addData("Encoder Value", test_motor.getCurrentPosition());
        telemetry.addData("Status", "Running");
        telemetry.update();

    }
```

> Para obter mais informações sobre a programação de encoders, consulte a página "Using Encoders" (Usando Encoders). Para obter mais informações sobre a métrica "counts per revolution" (contagens por revolução) e como usá-la, consulte a página "Encoders" (Encoders).

# Programando sensores

## Noções básicas do sensor de toque

O objetivo desta seção é abordar alguns dos conceitos básicos de programação de um dispositivo digital, ou sensor de toque, dentro do ambiente de programação Blocks.

> Antes de programar com um sensor de toque ou outro dispositivo digital, é importante entender o que é um dispositivo digital e quais são as aplicações comuns para esses dispositivos. Visite a página "Digital Sensors" (Sensores Digitais) para obter mais informações.

## Programando um dispositivo digital

> Modifique seu modo operacional para adicionar o código relacionado ao dispositivo digital. Isso pode ser feito limpando as modificações de código atuais ou adicionando o código do dispositivo digital ao seu modo operacional.

A informação proveniente de dispositivos digitais vem em dois estados, também conhecidos como estados binários. A forma mais comum de utilizar essa informação é através de uma instrução condicional, como uma instrução if/else. A linha test_touch.getState(); coleta o estado binário FALSO/VERDADEIRO do sensor de toque e atua como condição para a instrução if/else.

```
if (test_touch.getState()){
   //Touch Sensor is not pressed
} else {
   //Touch Sensor is pressed
      }
```

O código acima destaca a estrutura básica da instrução if/else para um dispositivo digital. O estado FALSO/VERDADEIRO de um sensor de toque REV corresponde a se o botão no sensor de toque está pressionado ou não. Quando o botão não está pressionado, o estado do sensor de toque é verdadeiro. Quando o botão é pressionado, o estado do sensor de toque é falso. Essa condição é refletida pelos comentários no código.

A maneira mais básica de usar um dispositivo digital é usar a telemetria para exibir informações, como o status do botão do sensor de toque. Para fazer isso, vamos criar uma variável de string chamada touchStatus. Essa variável será criada dentro do modo de operação (op mode).

> String é um conjunto de caracteres

```
public void runOpMode() {
      imu = hardwareMap.get(Gyroscope.class, "imu");
      test_color = hardwareMap.get(ColorSensor.class, "test_color");
      test_motor = hardwareMap.get(DcMotor.class, "test_motor");
      test_servo = hardwareMap.get(Servo.class, "test_servo");
      test_touch = hardwareMap.get(DigitalChannel.class, "test_touch");


      String touchStatus = "";
```

A linha String touchStatus = ""; declara que a variável touchStatus é uma variável de string vazia. Isso significa que touchStatus está atualmente armazenando uma string sem nenhum caractere.

```
public void runOpMode() {
      imu = hardwareMap.get(Gyroscope.class, "imu");
      test_color = hardwareMap.get(ColorSensor.class, "test_color");
      test_motor = hardwareMap.get(DcMotor.class, "test_motor");
      test_servo = hardwareMap.get(Servo.class, "test_servo");
      test_touch = hardwareMap.get(DigitalChannel.class, "test_touch");
```

```
        String touchStatus = "";

        telemetry.addData("Status", "Initialized");
        telemetry.update();

        // Wait for the game to start (driver presses PLAY)
        waitForStart();

        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {

            if (test_touch.getState()){
                //Touch Sensor is not pressed
            } else {
                //Touch Sensor is pressed
                        }
            telemetry.addData("Status", "Running");
            telemetry.update();

        }
    }
}
```

Agora, a variável touchStatus está vazia, mas para este exemplo, ela deverá ser alterada para refletir o status do sensor de toque. Para fazer isso, touchStatus deve ser definida como "Não Pressionado" ou "Pressionado".

```
if (test_touch.getState()){
    //Touch Sensor is not pressed
    touchStatus = "Not Pressed";

} else {
    //Touch Sensor is pressed
    touchStatus = "Pressed";
                    }
```

Para exibir as informações atribuídas ao touchStatus, é necessário utilizar a telemetria. Na seção de fundamentos de programação, a linha telemetry.addData() foi discutida brevemente. Essa chamada de método recebe um parâmetro de chave e variável, e exibe as informações no Driver Station. A chave é uma string, ou uma linha de texto, que deve definir a variável. Neste caso, a telemetry.addData() está sendo usada para exibir alterações na variável touchStatus, então "Touch

Status" seria uma boa chave. O parâmetro será a variável touchStatus. Adicione esta linha acima da linha telemetry.update(); no loop while.

```
telemetry.addData("Touch Sensor", touchStatus);
```

# Dispositivos digitais como interruptores de limite

Um dos usos mais comuns para um dispositivo digital, como um sensor de toque, é utilizá-lo como um interruptor de limite. A finalidade de um interruptor de limite é interromper um mecanismo, como um braço ou elevador, antes que ele ultrapasse suas limitações físicas. Nesta aplicação, a alimentação precisa ser cortada do motor quando o limite é atingido. Programar um interruptor de limite requer a mesma lógica if/else aplicada na seção anterior. Se o estado do sensor de toque for verdadeiro (não está pressionado), o motor terá energia. Caso contrário (está pressionado), o motor não terá energia.

```
 if (test_touch.getState()){
    //Touch Sensor is not pressed
     test_motor.setPower(0.3);

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

O bloco de código acima introduz os conceitos básicos de um interruptor de limite. Assim como na maioria dos sensores, é bom ter telemetria que atualiza a Estação do Motorista sobre o status do sensor. Considere o seguinte código:

```
public void runOpMode() {
        imu = hardwareMap.get(Gyroscope.class, "imu");
        test_color = hardwareMap.get(ColorSensor.class, "test_color");
        test_motor = hardwareMap.get(DcMotor.class, "test_motor");
        test_servo = hardwareMap.get(Servo.class, "test_servo");
        test_touch = hardwareMap.get(DigitalChannel.class, "test_touch");

        String touchStatus = "";

        telemetry.addData("Status", "Initialized");
        telemetry.update();
```

```
        // Wait for the game to start (driver presses PLAY)
        waitForStart();

        // run until the end of the match (driver presses STOP)
        while (opModeIsActive()) {

         if (test_touch.getState()){
            //Touch Sensor is not pressed
           test_motor.setPower(0.3);
           touchStatus = "Not Pressed";

          } else {
            //Touch Sensor is pressed
            test_motor.setPower(0);
            touchStatus = "Pressed";
                      }

            telemetry.addData("Touch Sensor:", touchStatus);
            telemetry.addData("Status", "Running");
            telemetry.update();

        }
    }
}
```

Revisão #1
Criado 18 dezembro 2023 16:17:11 por Enzo Coutinho
Atualizado 18 dezembro 2023 17:41:46 por Enzo Coutinho