Programação

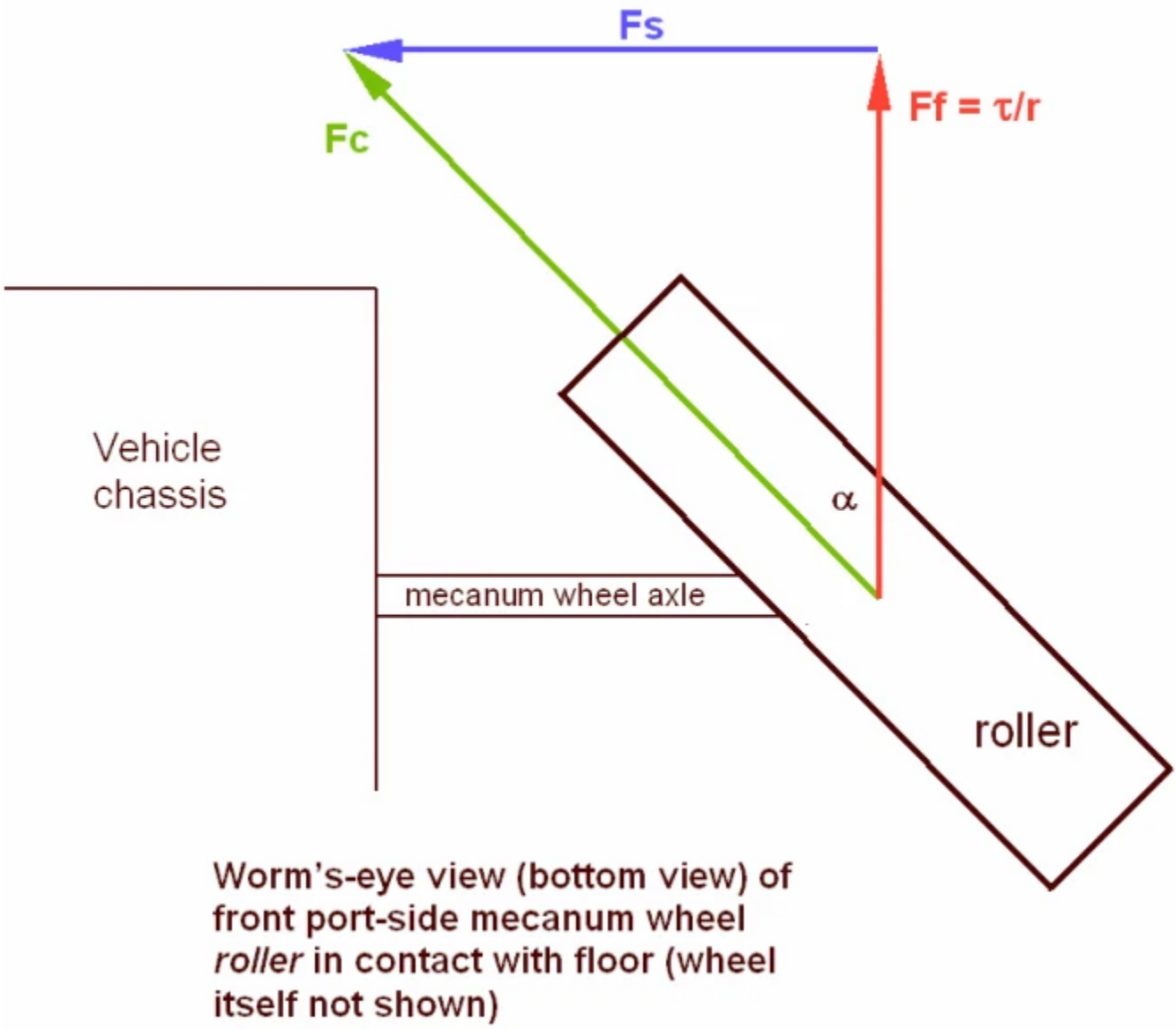
- Cinemática da Mecanum
- Orientação ao campo

Cinemática da Mecanum

O sistema de tração Mecanum é um tipo de sistema de tração muito popular na FTC®. Os sistemas de tração Mecanum permitem movimentos holonômicos. Isso significa que o sistema de tração é capaz de se mover em qualquer direção enquanto gira: para frente, para trás, de lado, transladando enquanto gira, etc. Aqui está um vídeo interessante demonstrando esse tipo de movimento.

> Alguns dos COTS mecanum disponíveis: goBILDA Strafer Chassis Kit e REV Mecanum Drivetrain Kit.

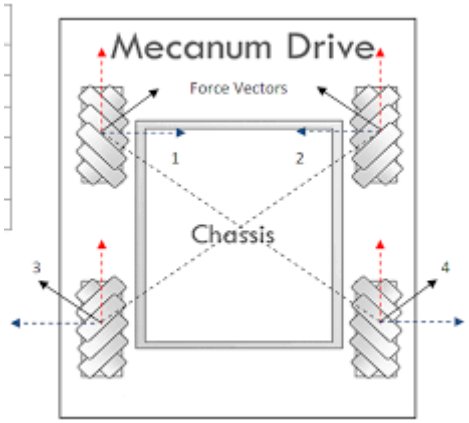
As rodas Mecanum possuem roletes em um ângulo de 45° em relação ao restante da roda. Como eles estão em contato com o solo em vez de algo sólido, como em uma roda de tração, em vez de a roda criar uma força paralela à sua orientação, ela gera uma força em um ângulo de 45° em relação à paralela. Dependendo de como as rodas são acionadas, os componentes X ou Y dos vetores de força podem se anular, permitindo o movimento em qualquer direção.



Worm's-eye view (bottom view) of front port-side mecanum wheel *roller* in contact with floor (wheel itself not shown)

Usando vetores para criar movimentos omnidirecionais

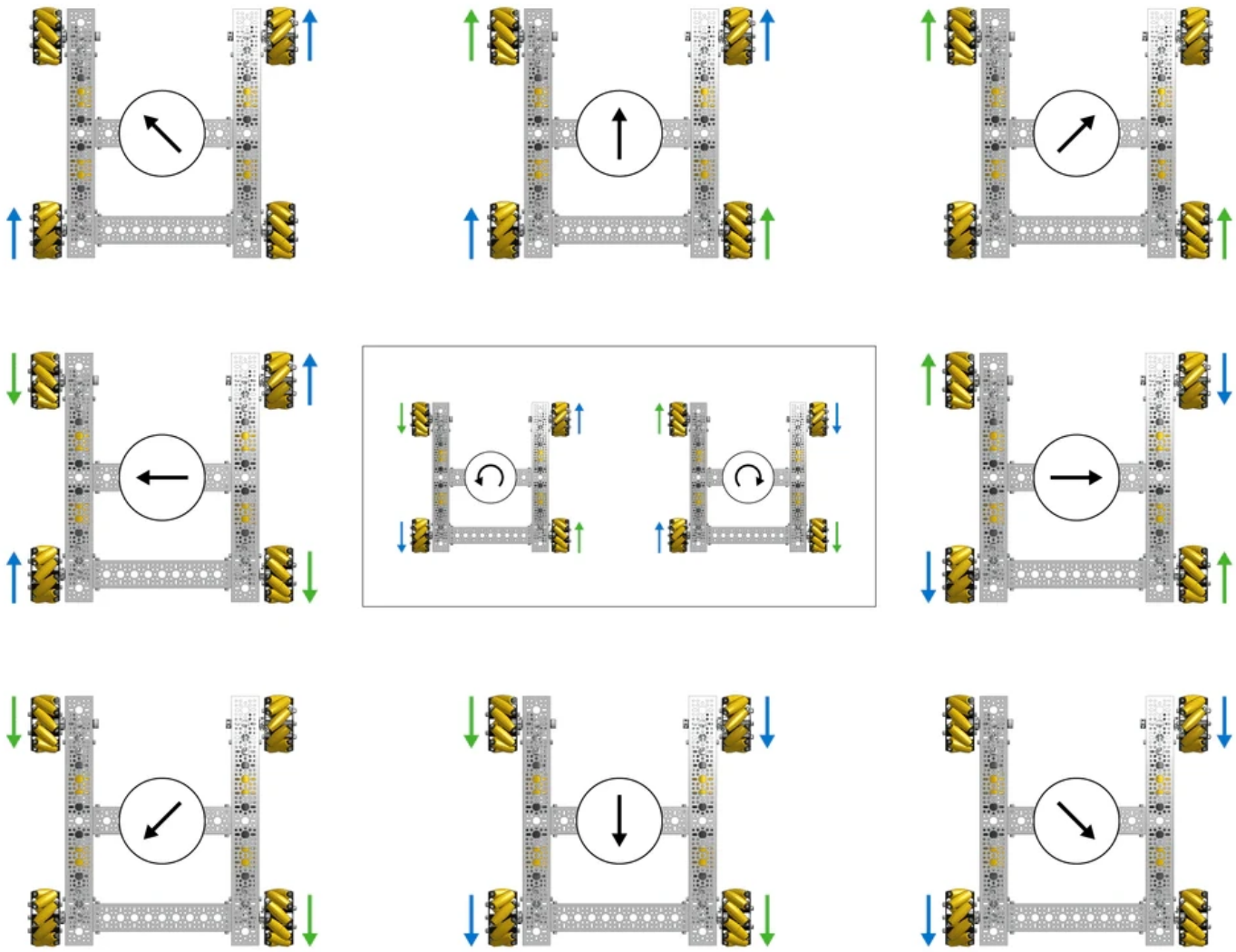
Uma configuração padrão de tração Mecanum possui 4 rodas Mecanum orientadas em forma de "X". Isso significa que os roletes estão inclinados em direção ao centro quando vistos de cima. Essa configuração permite somar os vetores de força gerados pelos roletes deslocados e derivar o movimento em qualquer direção. É importante notar que, devido ao atrito, o movimento perfeito não é possível em todas as direções. Assim, um sistema de tração Mecanum será capaz de se mover ligeiramente mais rápido para frente/trás do que em outras direções. Combinar translação e rotação também resultará em um movimento mais lento.



Na imagem acima, os vetores 1, 2, 3 e 4 representam os vetores de força criados pelas rodas Mecanum quando o chassi é movimentado para o topo da imagem. Todos os motores estão dirigindo para frente. As linhas azuis e vermelhas representam seus componentes X e Y, respectivamente. Aqui estão alguns exemplos de como as rodas devem ser acionadas para alcançar diferentes movimentos:

goBILDA®
DREAM • MAKE • REPEAT

PRODUCT INSIGHTS



É altamente recomendado não codificar rigidamente esses movimentos; existe uma maneira muito melhor descrita abaixo, que permite um movimento verdadeiramente holonômico e é muito mais elegante.

Derivando a equação de movimento da Mecanum

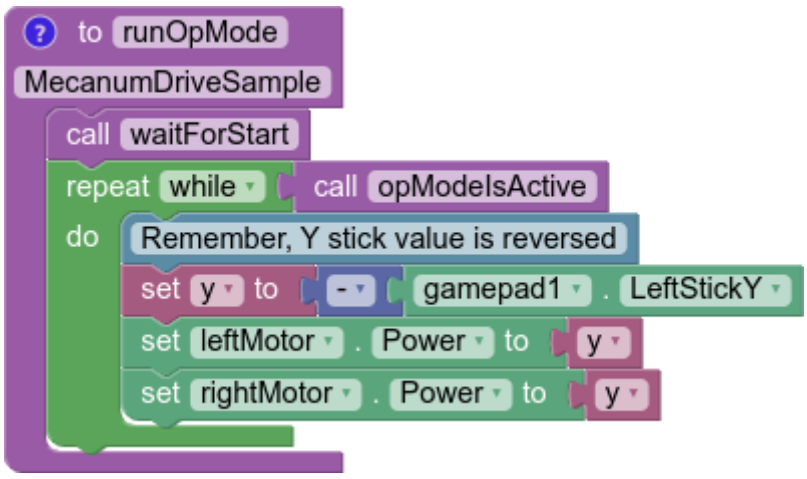
Antes de pensar em um sistema Mecanum, imagine um cenário em que você possui um sistema de tração tipo tanque com 2 motores, que deseja controlar usando o eixo Y do analógico esquerdo para movimentos para frente/trás e o eixo X do analógico direito para giros pivotantes. Os motores estão configurados de modo que o movimento positivo seja no sentido horário para o motor direito, quando o corpo do robô está voltado para longe de você, e o motor esquerdo seja o oposto. Para controlar apenas o movimento para frente/trás, basta definir as potências dos motores com base no valor do eixo Y do analógico (invertendo o sinal, já que Y é reverso):

Java

```
double y = -gamepad1.left_stick_y; // Remember, Y stick is reversed!


leftMotor.setPower(y);
rightMotor.setPower(y);
```

Blocks



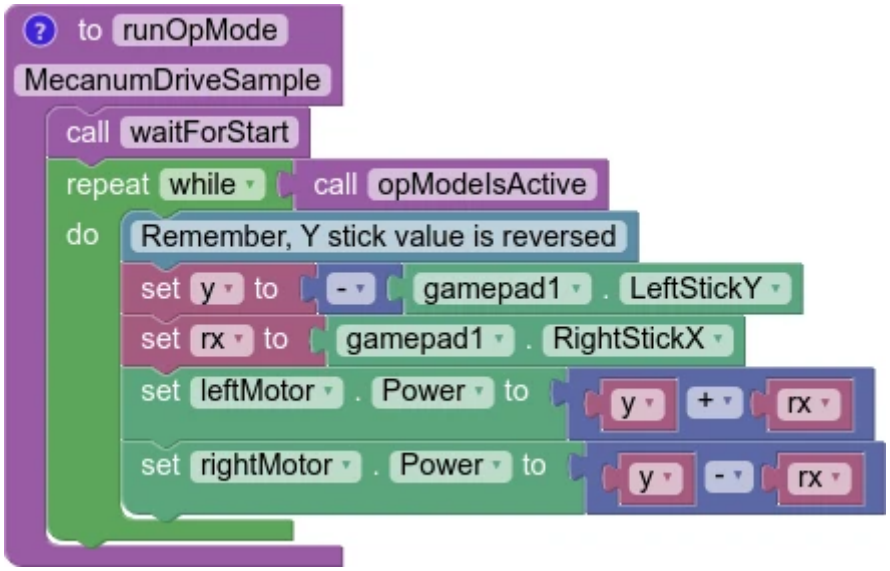
Embora inicialmente adicionar rotação possa parecer uma tarefa difícil, na verdade é super simples. Tudo o que você precisa fazer é subtrair o valor do eixo X do analógico direito das rodas direitas e adicioná-lo às rodas esquerdas:

Java

```
double y = -gamepad1.left_stick_y; // Remember, Y stick is reversed!
double rx = gamepad1.right_stick_x;


leftMotor.setPower(y + rx);
rightMotor.setPower(y - rx);
```

Blocks



Aqui, se o analógico esquerdo for pressionado para cima, ambos os motores receberão um valor positivo, fazendo o robô se mover para frente. Se for pressionado para baixo, ambos os motores receberão um valor negativo, fazendo o robô se mover para trás. Um princípio semelhante se aplica à rotação: se o analógico direito for empurrado para a direita, as rodas esquerdas girarão para frente enquanto as direitas girarão para trás, causando rotação. O oposto acontece ao empurrar o analógico para a esquerda. Se ambos os sticks forem pressionados ao mesmo tempo, por exemplo, com o eixo Y do analógico esquerdo em 1 e o eixo X do analógico direito também em 1, o valor das rodas esquerdas será 1 + 1 = 2 (que é limitado a 1 no SDK), e o das rodas direitas será 1 - 1 = 0, o que resulta em uma curva para a direita.

Aplicar movimento omnidirecional com rodas Mecanum segue o mesmo princípio de adicionar rotação no exemplo de tração tipo tanque. Os valores do eixo X do analógico esquerdo serão adicionados ou subtraídos a cada roda, dependendo de como essa roda precisa girar para alcançar o movimento desejado. A única diferença em relação à rotação é que, em vez de as rodas do mesmo lado terem o mesmo sinal, as rodas diagonais entre si terão o mesmo sinal.

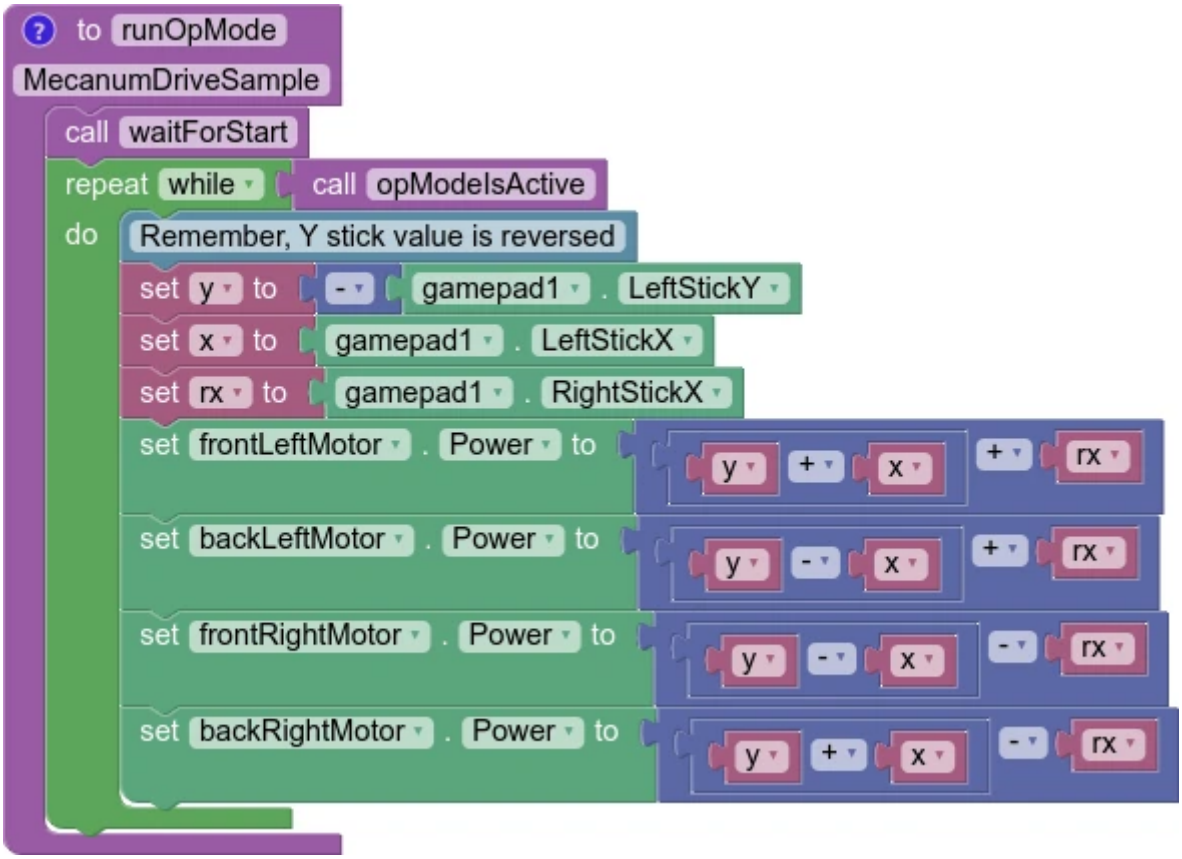
Queremos que um valor positivo no eixo X do analógico esquerdo corresponda a um movimento lateral para a direita. Referindo-nos à imagem de vetores, isso significa que as rodas dianteira esquerda e traseira direita precisam girar para frente, enquanto as rodas traseira esquerda e dianteira direita precisam girar para trás. Portanto, devemos adicionar o valor de X às rodas dianteira esquerda e traseira direita e subtraí-lo das rodas traseira esquerda e dianteira direita.

Java

```
double y = -gamepad1.left_stick_y; // Remember, Y stick is reversed!
double x = gamepad1.left_stick_x;
double rx = gamepad1.right_stick_x;

frontLeftMotor.setPower(y + x + rx);
backLeftMotor.setPower(y - x + rx);
frontRightMotor.setPower(y - x - rx);
backRightMotor.setPower(y + x - rx);
```

Blocks



> A maioria dos motores FTC gira no sentido anti-horário quando vistos de frente, quando recebem potência positiva por padrão, com exceção dos NeveRests. Se o seu sistema de tração usa um número par de engrenagens, isso inverterá a direção em que os motores giram. Na maioria dos sistemas de tração, será necessário inverter o lado esquerdo para que a potência positiva mova o robô para frente com a maioria dos motores, e inverter o lado direito com os NeveRests. A presença de engrenagens entre a caixa de engrenagens do motor e a roda pode inverter isso, como é o caso do goBILDA Strafer e do Kit de Tração Mecanum REV.

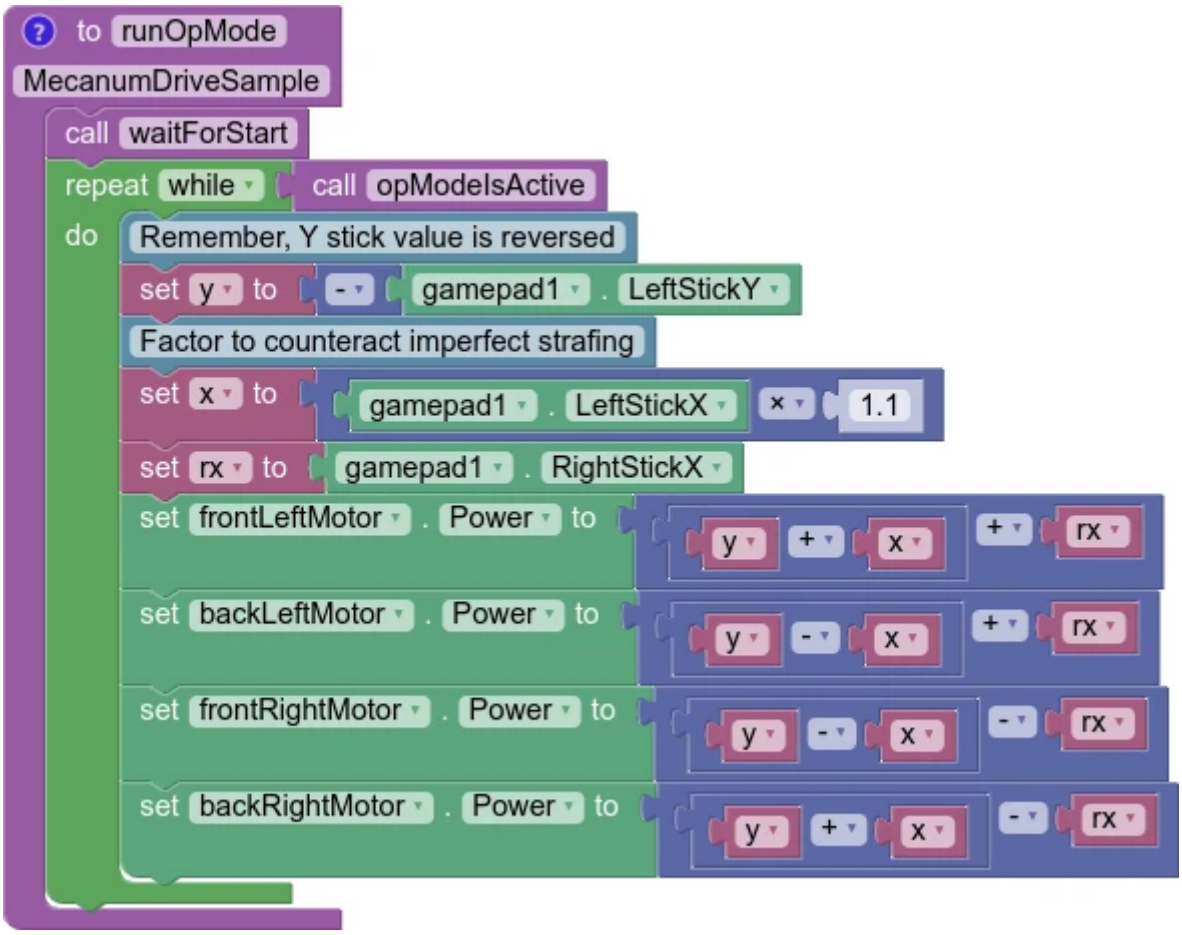
Isso é o mesmo que o exemplo do tanque, exceto agora com 4 motores e o componente de "strafing" (movimento lateral) adicionado. Semelhante ao exemplo do tanque, o componente Y é adicionado a todas as rodas, e o X direito (rx) é adicionado às rodas esquerdas e subtraído das rodas direitas. Agora, adicionamos um componente de X esquerdo (x) que nos permite fazer o "strafing" para a direita. Ao fazer isso, no entanto, permitimos o "strafing" em qualquer direção. Se você pensar bem, pressionar o joystick esquerdo para a esquerda fará a mesma coisa, mas ao contrário, o que é necessário para o "strafing" para a esquerda. Se ele for pressionado em 45 graus, os componentes x e y do joystick serão iguais. Isso fará com que dois motores diagonais se cancelem, permitindo o movimento diagonal. Esse mesmo efeito se aplica a qualquer ângulo do joystick.

Agora que temos um programa de condução Mecanum funcional, há algumas coisas que podem ser feitas para aprimorá-lo. A primeira delas seria multiplicar o valor de X esquerdo por algo para corrigir o "strafing" imperfeito. Fazer isso fará com que a condução seja mais precisa em direções não alinhadas aos eixos e tornará a condução centrada no campo mais precisa. Neste tutorial, usaremos 1.1, mas isso realmente depende da preferência do motorista.

Java

```
double y = -gamepad1.left_stick_y; // Remember, Y stick is reversed!
double x = gamepad1.left_stick_x * 1.1; // Counteract imperfect strafing
double rx = gamepad1.right_stick_x;
```

Blocks



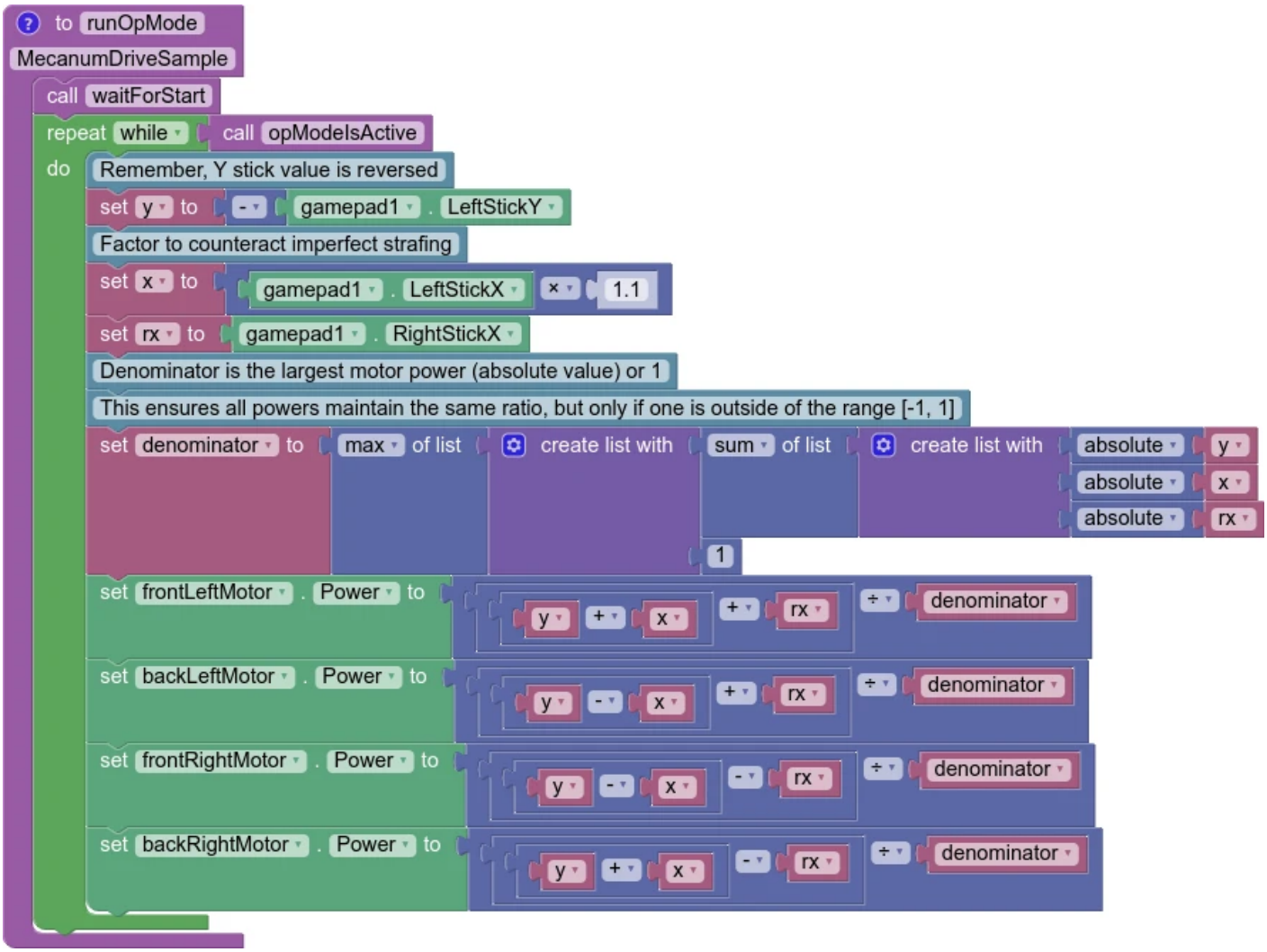
A outra melhoria que podemos fazer é escalar os valores para o intervalo de -1 a 1.

Como o SDK simplesmente limita (clipe) as potências para esse intervalo, podemos perder a proporção que estamos buscando, a menos que proativamente coloquemos todos os números de volta nesse intervalo, mantendo ainda nossa proporção calculada. Por exemplo, se calcularmos os valores de 0,4, 0,1, 1,1 e 1,4, eles serão limitados para 0,4, 0,1, 1,0 e 1,0, o que não é a mesma proporção. Em vez disso, precisamos dividir todos eles pelo valor absoluto da maior potência quando ela exceder 1:

Java

```
// Denominator is the largest motor power (absolute value) or 1
// This ensures all the powers maintain the same ratio, but only when
// at least one is out of the range [-1, 1]
double denominator = Math.max(Math.abs(y) + Math.abs(x) + Math.abs(rx), 1);
double frontLeftPower = (y + x + rx) / denominator;
double backLeftPower = (y - x + rx) / denominator;
double frontRightPower = (y - x - rx) / denominator;
double backRightPower = (y + x - rx) / denominator;
```

Blocks



Certifique-se de definir as potências dos motores e atualiza-lás a cada ciclo do opmode!

Código de exemplo

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;
```

```
import com.qualcomm.robotcore.hardware.DcMotorSimple;

@TeleOp
public class MecanumTeleOp extends LinearOpMode {
    @Override
    public void runOpMode() throws InterruptedException {
        // Declare our motors
        // Make sure your ID's match your configuration
        DcMotor frontLeftMotor = hardwareMap.dcMotor.get("frontLeftMotor");
        DcMotor backLeftMotor = hardwareMap.dcMotor.get("backLeftMotor");
        DcMotor frontRightMotor = hardwareMap.dcMotor.get("frontRightMotor");
        DcMotor backRightMotor = hardwareMap.dcMotor.get("backRightMotor");

        // Reverse the right side motors. This may be wrong for your setup.
        // If your robot moves backwards when commanded to go forwards,
        // reverse the left side instead.
        // See the note about this earlier on this page.
        frontRightMotor.setDirection(DcMotorSimple.Direction.REVERSE);
        backRightMotor.setDirection(DcMotorSimple.Direction.REVERSE);

        waitForStart();

        if (isStopRequested()) return;

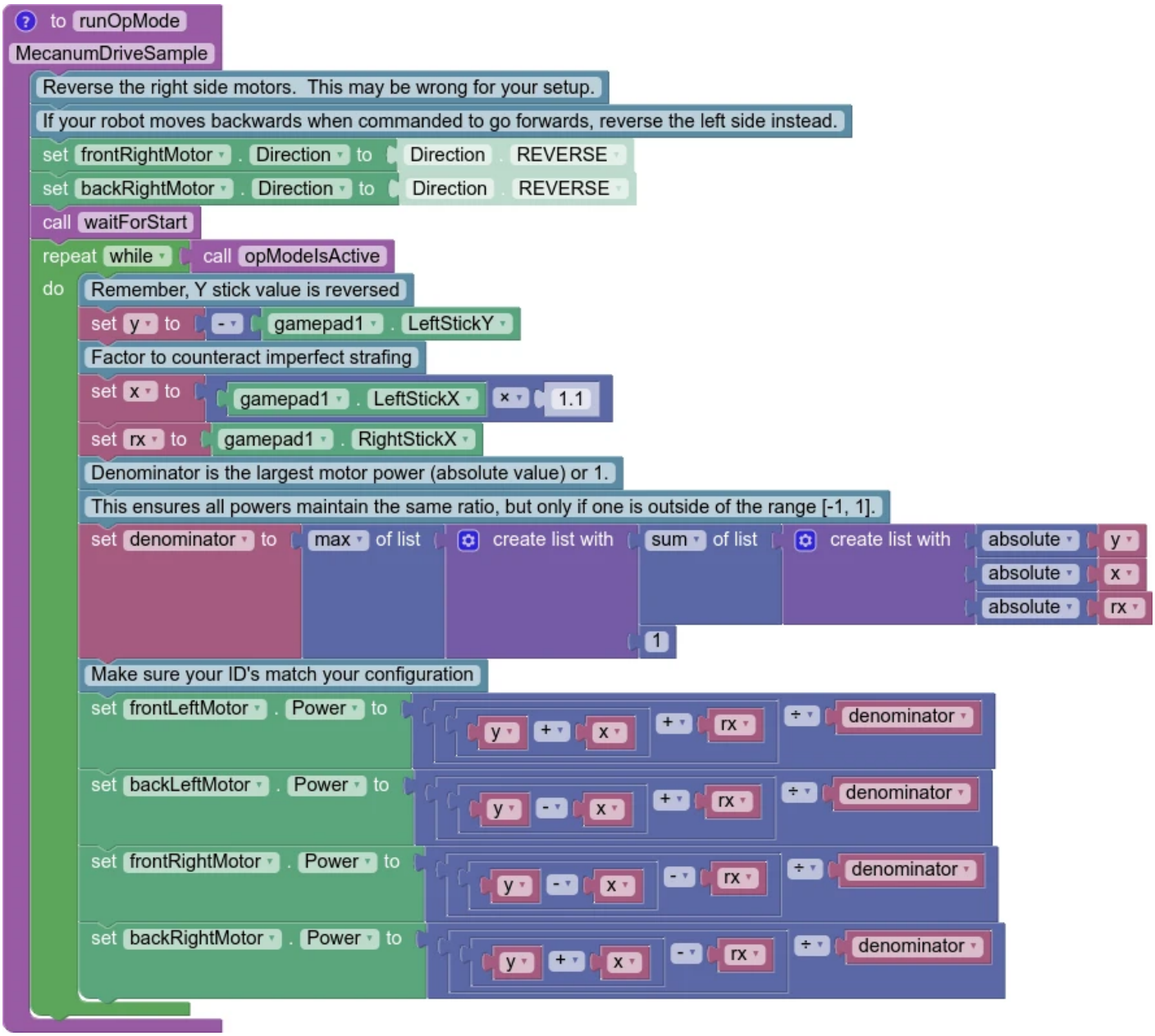
        while (opModeIsActive()) {
            double y = -gamepad1.left_stick_y; // Remember, Y stick value is reversed
            double x = gamepad1.left_stick_x * 1.1; // Counteract imperfect strafing
            double rx = gamepad1.right_stick_x;

            // Denominator is the largest motor power (absolute value) or 1
            // This ensures all the powers maintain the same ratio,
            // but only if at least one is out of the range [-1, 1]
            double denominator = Math.max(Math.abs(y) + Math.abs(x) + Math.abs(rx), 1);
            double frontLeftPower = (y + x + rx) / denominator;
            double backLeftPower = (y - x + rx) / denominator;
            double frontRightPower = (y - x - rx) / denominator;
            double backRightPower = (y + x - rx) / denominator;

            frontLeftMotor.setPower(frontLeftPower);
            backLeftMotor.setPower(backLeftPower);
```

```
            frontRightMotor.setPower(frontRightPower);
            backRightMotor.setPower(backRightPower);
        }
    }
}
```

Blocks



Próximos passos

Caso queira aumentar o nível de movimentação da mecanum, faça-o ser centralizado ao campo, veja como na próxima página.

Orientação ao campo

Com a condução Mecanum centrada no campo, o joystick de translação controla a direção do robô em relação ao campo, em vez do quadro do robô. Isso é preferido por alguns *drivers* e torna algumas ações evasivas mais fáceis, pois é possível girar enquanto se translaciona em uma direção específica. Para fazer isso, os componentes x/y dos joysticks são rotacionados no sentido oposto ao ângulo do robô, que é fornecido pelo IMU.

Há um IMU dentro dos Control Hubs (e modelos mais antigos de Expansion Hubs). Diferente de outros hardwares, é recomendado fazer mais do que apenas usar hardwareMap.get() para começar a usá-lo. Vale notar que, por padrão, ele é configurado como "imu" ao criar uma nova configuração.

Consulte a página de documentação do FTC sobre a interface IMU para mais informações. A maneira como o IMU será inicializado aqui é:

```
// Retrieve the IMU from the hardware map
imu = hardwareMap.get(IMU.class, "imu");
// Adjust the orientation parameters to match your robot
IMU.Parameters parameters = new IMU.Parameters(new RevHubOrientationOnRobot(
        RevHubOrientationOnRobot.LogoFacingDirection.UP,
        RevHubOrientationOnRobot.UsbFacingDirection.FORWARD));
// Without this, the REV Hub's orientation is assumed to be logo up / USB forward
imu.initialize(parameters);
```

O ângulo precisa ser lido a cada loop. Além disso, embora o IMU mantenha uma posição zero consistente entre os OpModes (incluindo entre autônomo e teleop), adicionar uma função para redefinir o ângulo é importante para corrigir o erro e porque o zero pode mudar devido a alguns tipos de desconexões.

> Objetos BNO055 redefinirão o zero do IMU quando o método initialize for chamado. A classe BNO055 não é recomendada para novos desenvolvimentos. A classe IMU não possui esse comportamento e é a substituição adequada a partir da versão SDK v8.1.

```
// This button choice was made so that it is hard to hit on accident,
// it can be freely changed based on preference.
// The equivalent button is start on Xbox-style controllers.
if (gamepad1.options) {
    imu.resetYaw();
}
```

```
double botHeading = imu.getRobotYawPitchRollAngles().getYaw(AngleUnit.RADIANS);
```

Em seguida, os valores do joystick de translação precisam ser rotacionados no sentido oposto à rotação do robô. O IMU retorna o heading (ângulo de orientação), mas precisamos rotacionar o movimento no sentido oposto à rotação do robô, então utilizamos o valor negativo do ângulo. Os valores do joystick formam um vetor, e rotacionar um vetor em 2D requer a seguinte fórmula (provada aqui), onde:

- x1 e y1 são os componentes do vetor original;
- θ é o ângulo de rotação;
- x2 e y2 são os componentes do vetor resultante.

```
// Rotate the movement direction counter to the bot's rotation
double rotX = x * Math.cos(-botHeading) - y * Math.sin(-botHeading);
double rotY = x * Math.sin(-botHeading) + y * Math.cos(-botHeading);
```

Então, esses valores rotacionados podem ser inseridos na cinemática da Mecanum mostrada anteriormente.

```
double denominator = Math.max(Math.abs(rotY) + Math.abs(rotX) + Math.abs(rx), 1);
double frontLeftPower = (rotY + rotX + rx) / denominator;
double backLeftPower = (rotY - rotX + rx) / denominator;
double frontRightPower = (rotY - rotX - rx) / denominator;
double backRightPower = (rotY + rotX - rx) / denominator;
```

Código de exemplo

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

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.IMU;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorSimple;
import com.qualcomm.hardware.rev.RevHubOrientationOnRobot;
import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;

@TeleOp
public class FieldCentricMecanumTeleOp extends LinearOpMode {
```

```
@Override
public void runOpMode() throws InterruptedException {
    // Declare our motors
    // Make sure your ID's match your configuration
    DcMotor frontLeftMotor = hardwareMap.dcMotor.get("frontLeftMotor");
    DcMotor backLeftMotor = hardwareMap.dcMotor.get("backLeftMotor");
    DcMotor frontRightMotor = hardwareMap.dcMotor.get("frontRightMotor");
    DcMotor backRightMotor = hardwareMap.dcMotor.get("backRightMotor");

    // Reverse the right side motors. This may be wrong for your setup.
    // If your robot moves backwards when commanded to go forwards,
    // reverse the left side instead.
    // See the note about this earlier on this page.
    frontRightMotor.setDirection(DcMotorSimple.Direction.REVERSE);
    backRightMotor.setDirection(DcMotorSimple.Direction.REVERSE);

    // Retrieve the IMU from the hardware map
    IMU imu = hardwareMap.get(IMU.class, "imu");
    // Adjust the orientation parameters to match your robot
    IMU.Parameters parameters = new IMU.Parameters(new RevHubOrientationOnRobot(
            RevHubOrientationOnRobot.LogoFacingDirection.UP,
            RevHubOrientationOnRobot.UsbFacingDirection.FORWARD));
    // Without this, the REV Hub's orientation is assumed to be logo up / USB forward
    imu.initialize(parameters);

    waitForStart();

    if (isStopRequested()) return;

    while (opModeIsActive()) {
        double y = -gamepad1.left_stick_y; // Remember, Y stick value is reversed
        double x = gamepad1.left_stick_x;
        double rx = gamepad1.right_stick_x;

        // This button choice was made so that it is hard to hit on accident,
        // it can be freely changed based on preference.
        // The equivalent button is start on Xbox-style controllers.
        if (gamepad1.options) {
            imu.resetYaw();
        }
```

```
            double botHeading = imu.getRobotYawPitchRollAngles().getYaw(AngleUnit.RADIANS);

            // Rotate the movement direction counter to the bot's rotation
            double rotX = x * Math.cos(-botHeading) - y * Math.sin(-botHeading);
            double rotY = x * Math.sin(-botHeading) + y * Math.cos(-botHeading);

            rotX = rotX * 1.1;  // Counteract imperfect strafing

            // Denominator is the largest motor power (absolute value) or 1
            // This ensures all the powers maintain the same ratio,
            // but only if at least one is out of the range [-1, 1]
            double denominator = Math.max(Math.abs(rotY) + Math.abs(rotX) + Math.abs(rx), 1);
            double frontLeftPower = (rotY + rotX + rx) / denominator;
            double backLeftPower = (rotY - rotX + rx) / denominator;
            double frontRightPower = (rotY - rotX - rx) / denominator;
            double backRightPower = (rotY + rotX - rx) / denominator;

            frontLeftMotor.setPower(frontLeftPower);
            backLeftMotor.setPower(backLeftPower);
            frontRightMotor.setPower(frontRightPower);
            backRightMotor.setPower(backRightPower);
        }
    }
}
```