
GUÍA DE PROGRAMACIÓN
BY IMAGINEERS

1.ANTES DE PROGRAMAR

1.1.¿Qué hace exactamente la programación?

Antes de escribir código para su robot, es importante comprender qué es programar un robot. Cuando escribes código y lo subes a un robot, le estás dando a ese robot un conjunto de instrucciones que le indican cómo interactuar con un entorno. Piense en ello como enseñarle al robot a pensar y actuar. Sin el código, el robot sería sólo una máquina compuesta de motores, engranajes y sensores.

Es importante pensar en cómo funciona todo y abordar las cosas metódicamente para poder depurar cuando las cosas van mal y evitar que empeoren. También tienes que trabajar tu lógica y entender lo que escribes porque esto es fundamental.

Es obligatorio concentrarse en cómo trabajar con sus compañeros de equipo, cómo colaborar de manera eficiente y usar GitHub para compartir su código de una persona a otra. En el contexto de FTC es muy importante ser abiertos y colaborar. Además, como programador, tendrás que trabajar también con la gente del departamento de construcción, no sólo con los demás programadores.

Los robots son esencialmente máquinas que funcionan con señales electrónicas. Al programar, generalmente se crea una secuencia de impulsos o comandos que se transmiten a las distintas partes del robot. Los comandos deben estar en el orden correcto; le dicen al robot qué hacer y a qué hora. Cuando codifica, debe hacerlo paso a paso para que la depuración final sea fácil.

//EJEMPLOS

- Movimiento: Instrucciones para motores/servos que mueven las ruedas, extensiones u otras piezas móviles del robot.
- Sensores/Cámaras: Los sensores del robot recogen información sobre el entorno, como la distancia, el color o la orientación, y envían esa información al sistema de control.
- Comunicación: El robot puede enviarle información, como el nivel de la batería, la posición en el campo o el estado de los servos/motores.

//IMPORTANCIA

Saber que programar un robot significa crear relaciones de "causa y efecto" es el requisito básico para convertirse en un buen programador, va mucho más allá de escribir código. Al programar hay que pensar fuera de lo común y tratar de codificar cosas de manera eficiente. Diseña un sistema en el que todo lo que hará el robot tiene un propósito y, en definitiva, es útil. Antes de comenzar a codificar, debe conocer los principales efectos que obtendrá al escribir el código.

1.2.Los lenguajes de programación de FTC y la configuración de su espacio de trabajo.

En FTC, la mayoría de los equipos programan sus robots en Java. Por supuesto, siempre hay otra forma de codificar el robot, como bloques, pero resulta que Java es uno de los lenguajes de programación más utilizados debido a su solidez y confiabilidad, y porque es apropiado para la robótica. Al aprender Java, obtendrá control sobre su robot: podrá realizar tareas que impliquen movimiento, detección y toma de decisiones.

¿POR QUÉ JAVA?

Fácil de aprender: Java tiene una estructura clara, lo que lo hace amigable para principiantes y también es extenso, puedes usarlo para codificar una variedad de cosas, por lo que es muy útil aprender.

Ampliamente utilizado: El conocimiento de Java abre las puertas a muchas otras oportunidades de programación.

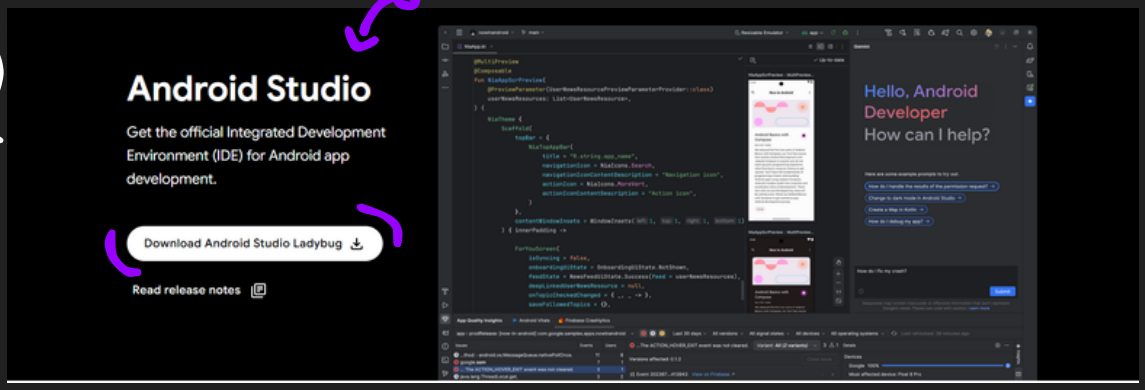
Soporte específico de FIRST Tech Challenge: El SDK oficial de FTC está escrito en Java, por lo que es el más fácil de usar al programar su robot. Además, la mayoría de los equipos eligen usar Java para que tenga más personas que puedan ayudarlos y asistirlos.

CARACTERÍSTICAS:

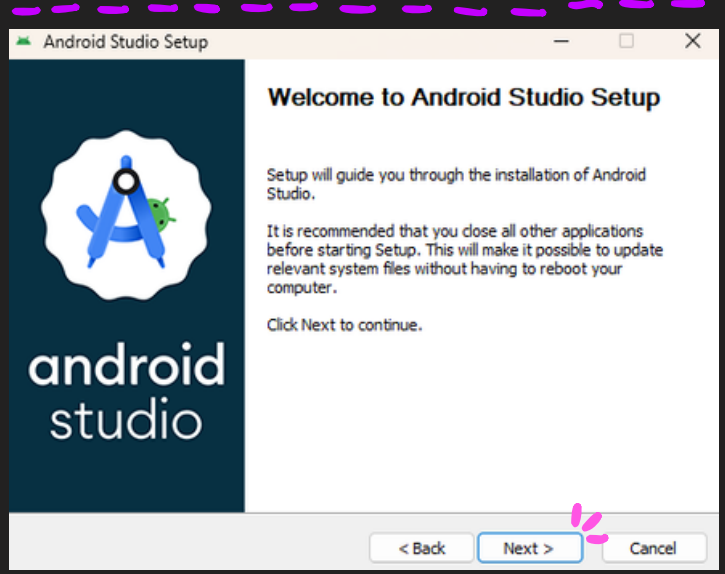
- Escriba código Java, con herramientas integradas, para ayudar a detectar errores.
- Simule las acciones de su robot para probar su código antes de ejecutarlo en el robot.
- Conéctese al SDK de FTC para conocer métodos prediseñados para operar su robot.

¿CÓMO INSTALAR?

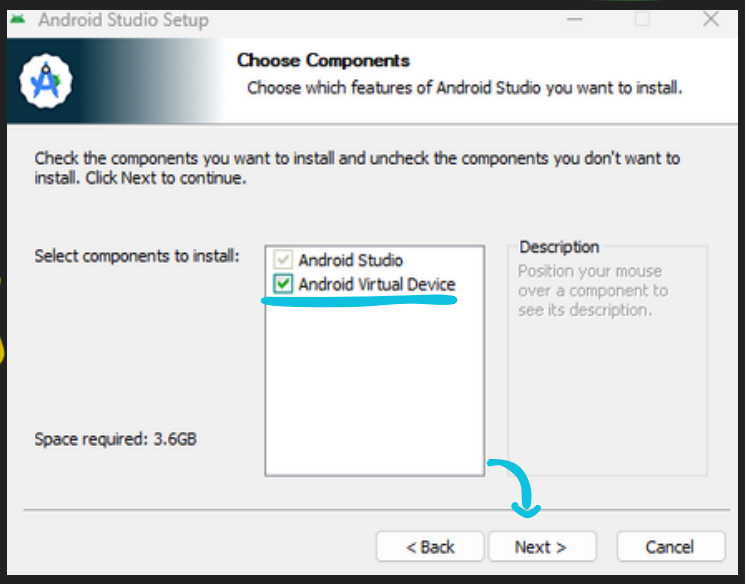
- Ir a Sitio web oficial de Android Studio y descarga la versión para tu sistema operativo.



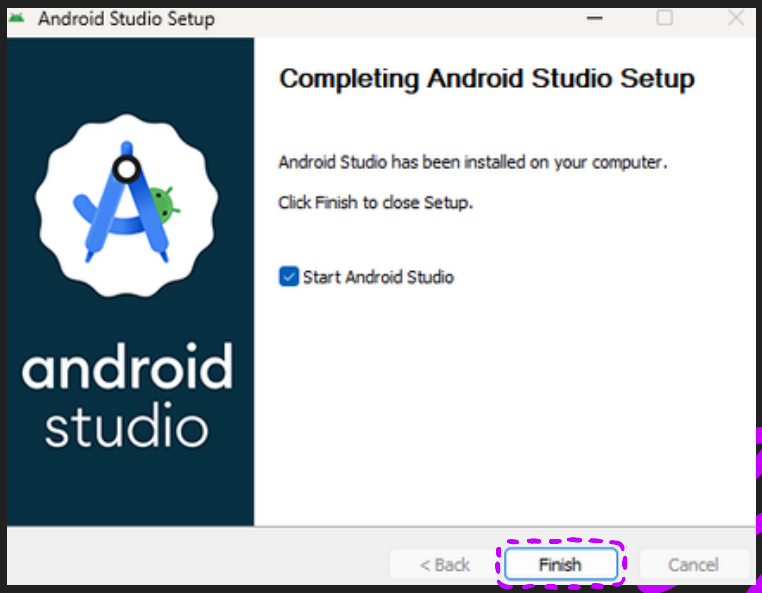
- Después de eso, abra el archivo ejecutable descargado y haga clic en Siguiente para iniciar la configuración.



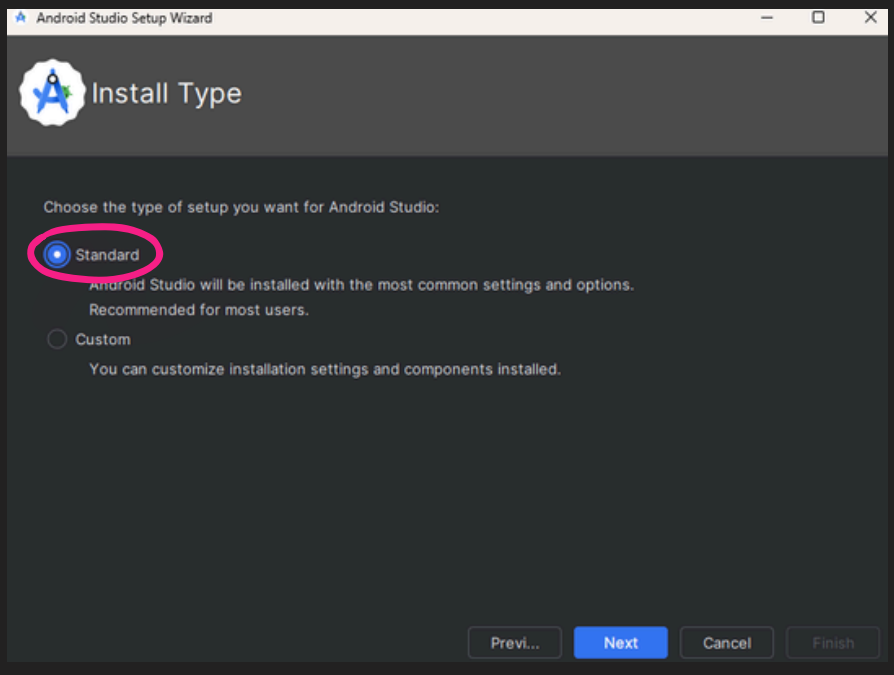
- Seleccione Android Studio y el dispositivo virtual Android y haga clic en Siguiente



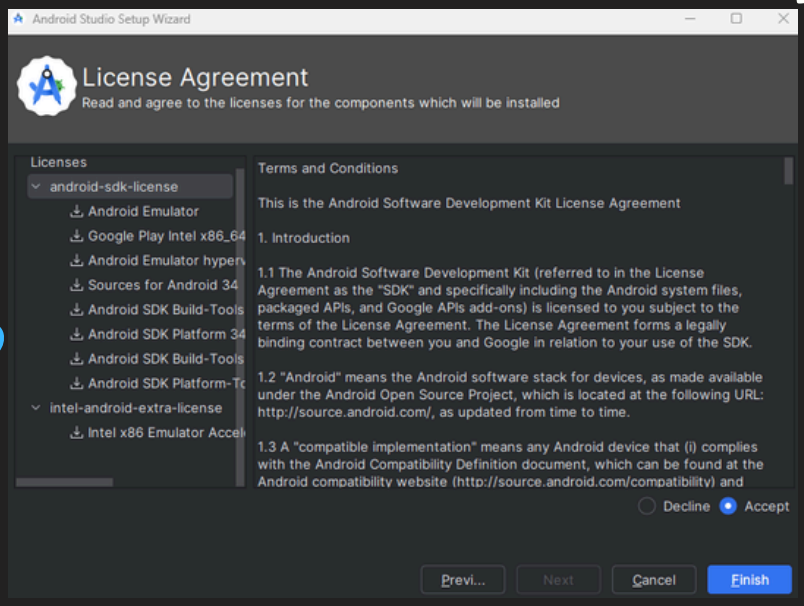
- Después de eso, haga clic en finalizar y abra el programa descargado.



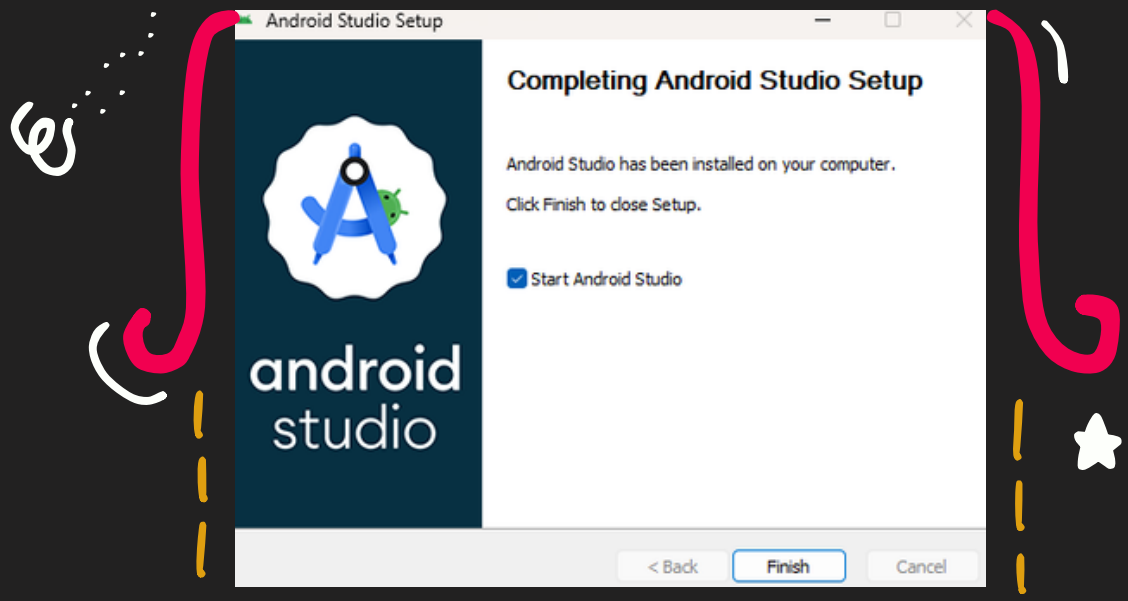
- Después de haber abierto el programa, seleccione la configuración estándar para Android Studio y haga clic en Siguiente.



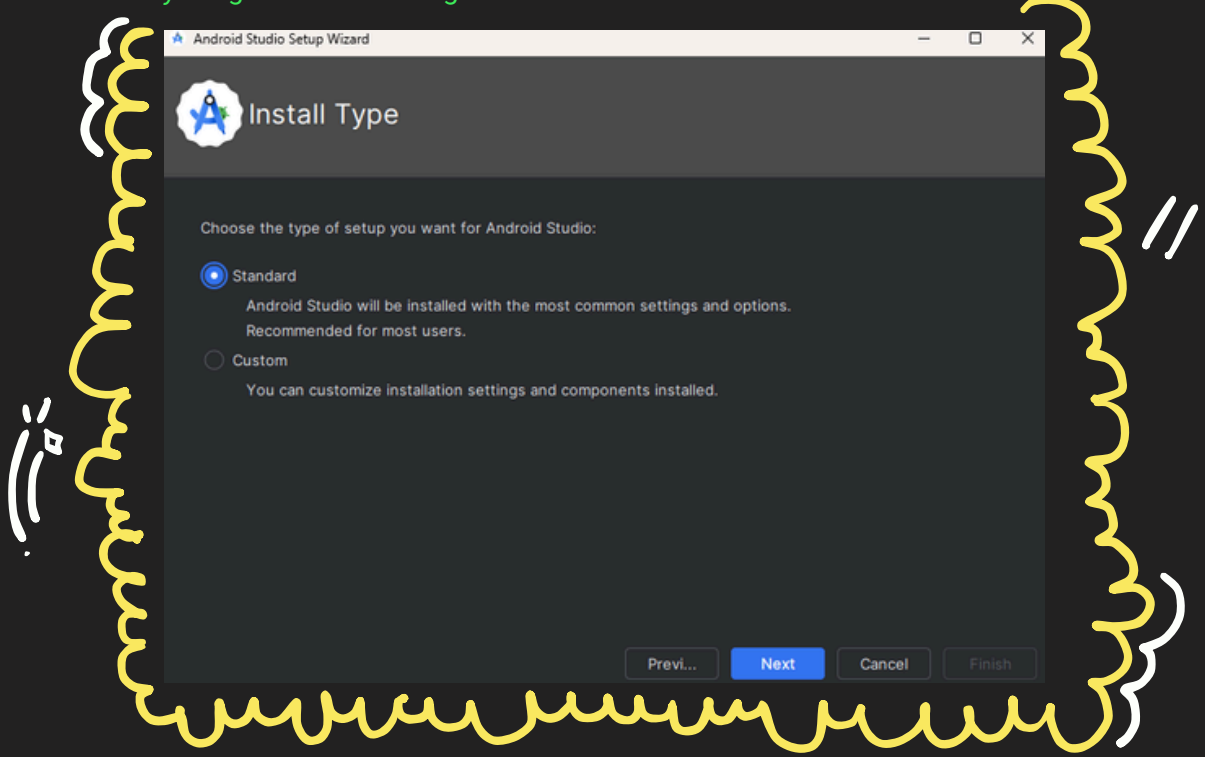
- Acepta los términos y condiciones de cada licencia y presiona siguiente



- Después de eso, haga clic en finalizar y abra el programa descargado.



- Después de haber abierto el programa, seleccione la configuración estándar para Android Studio y haga clic en Siguiente.

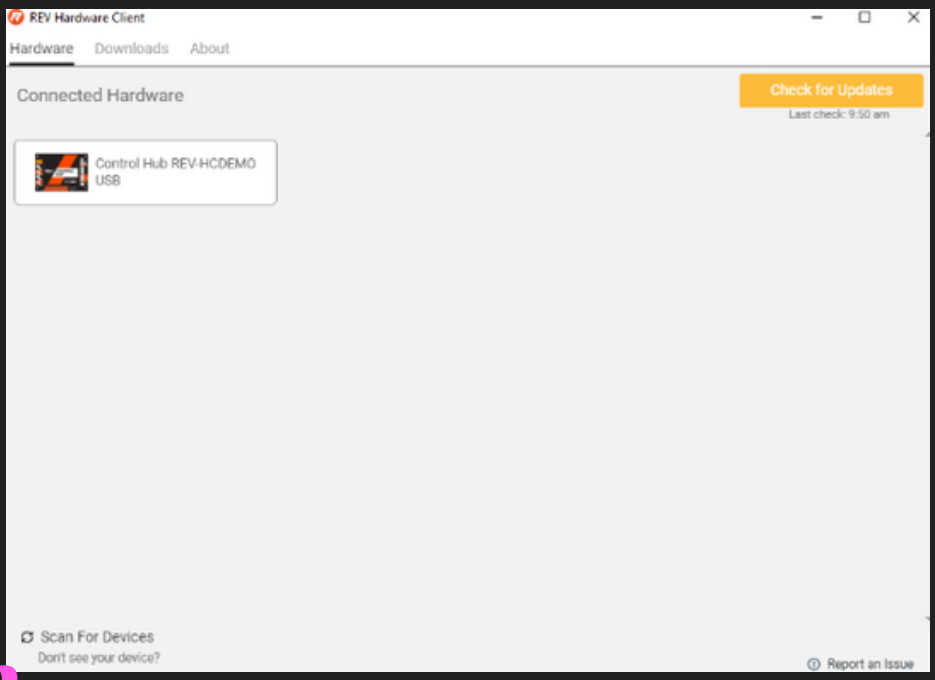


1.3.Preparación del centro de control

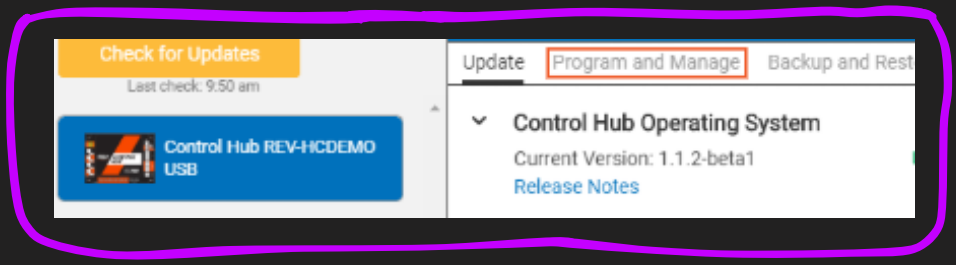
Para comenzar a configurar el Centro de control, en primer lugar, necesitarás una batería cargada para alimentarlo. Una vez que se enciende el Control Hub, la luz comenzará a brillar.

Después de eso, para poder configurarlo, debes instalar [Cliente de hardware Rev] desde su sitio. Para conectarse al Control Hub debe conectarse a su wifi. De forma predeterminada, el wifi del Control Hub tiene un nombre que comienza con "FTC-" o "FIRST-" seguido de cuatro caracteres que se asignan aleatoriamente. La contraseña predeterminada para la red es "contraseña".

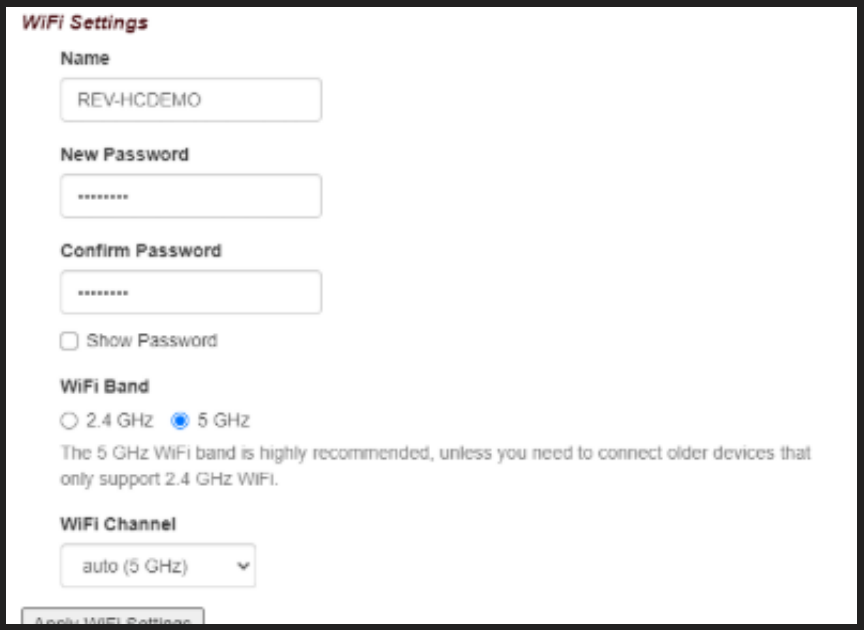
Una vez establecida la conexión, debe abrir el Rev Hardware Client recientemente descargado y verá algo como esto.

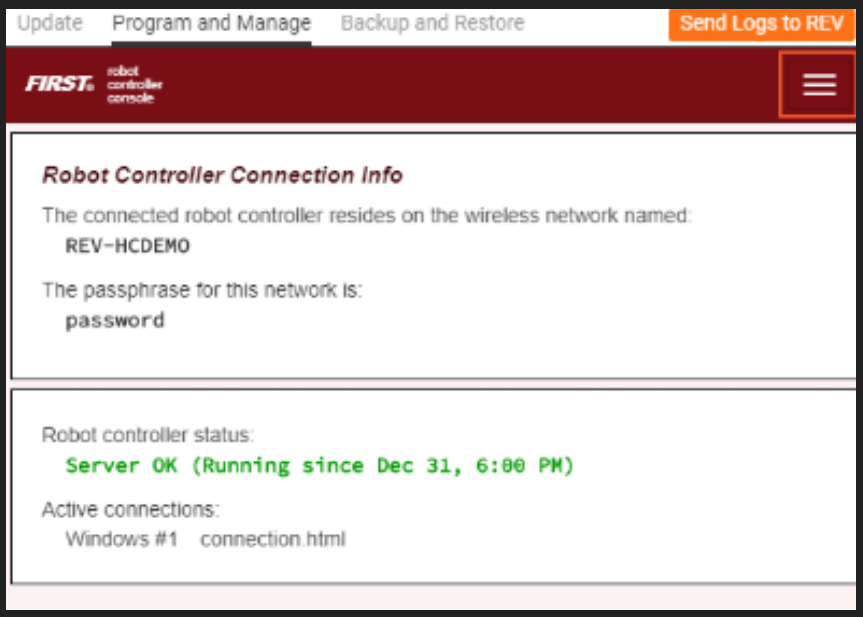


- Debe seleccionar Control Hub en la aplicación y hacer clic en el programa y administrar botón



- Ahora, en la parte superior derecha de la pantalla, habrá un menú de navegación donde podrás acceder a la configuración de wifi para cambiar la contraseña y el nombre del wifi.





- Para agregar un Hub de expansión, debe conectar el Hub de control y el Hub de expansión usando 2 cables, uno que conecta la batería y un cable de datos conectado a los puertos RS485 tanto en el Hub de control como en el Hub de expansión.

1.4.Instalación de la estación de controladores y mapeo de hardware

Una característica clave del SDK de la FTC, y de cualquier API de robótica, es proporcionar acceso de software al hardware físico del robot. Esto se hace a través de un proceso llamado mapeo de hardware, que permite a los programadores identificar e interactuar con dispositivos de hardware como motores, servos y sensores en su código.

- El mapeo de hardware implica dos pasos principales:
- Configuración de hardware en el dispositivo controlador del robot.
- Mapeo de hardware dentro de la clase OpMode de su programa.

//CONFIGURACIÓN DE HARDWARE

Cuando enciende el teléfono del controlador del robot y lo conecta al REV Expansion Hub, o cuando enciende el REV Control Hub, la aplicación del controlador utiliza el archivo de configuración de hardware activo para reconocer los dispositivos conectados. Este archivo de configuración le dice a la aplicación qué hardware (por ejemplo, motores, servos, sensores) está conectado al concentrador. La aplicación del controlador validará las conexiones y le notificará cualquier error si no puede comunicarse con los dispositivos configurados.

//ADMINISTRAR ARCHIVOS DE CONFIGURACIÓN

Puede crear varios archivos de configuración en el dispositivo controlador. Esto resulta útil si tiene diferentes configuraciones de hardware para el mismo robot o si el dispositivo controlador se utiliza con varios robots. Sin embargo, sólo puede haber un archivo de configuración activo a la vez. Se muestra el nombre de la configuración activa:

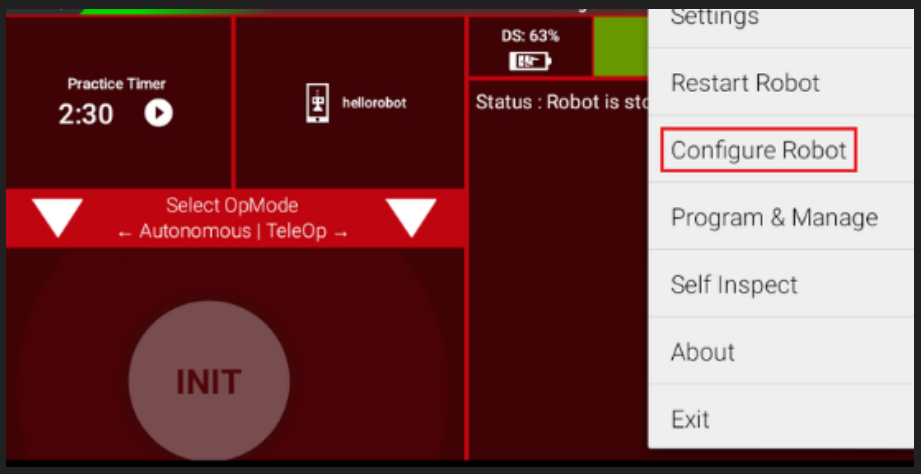
- En la esquina superior derecha de la pantalla de la aplicación del controlador del robot.
- En la pantalla de la estación del conductor para el centro de control.

//CREAR O EDITAR ARCHIVOS DE CONFIGURACIÓN

Para crear o acceder a archivos de configuración:

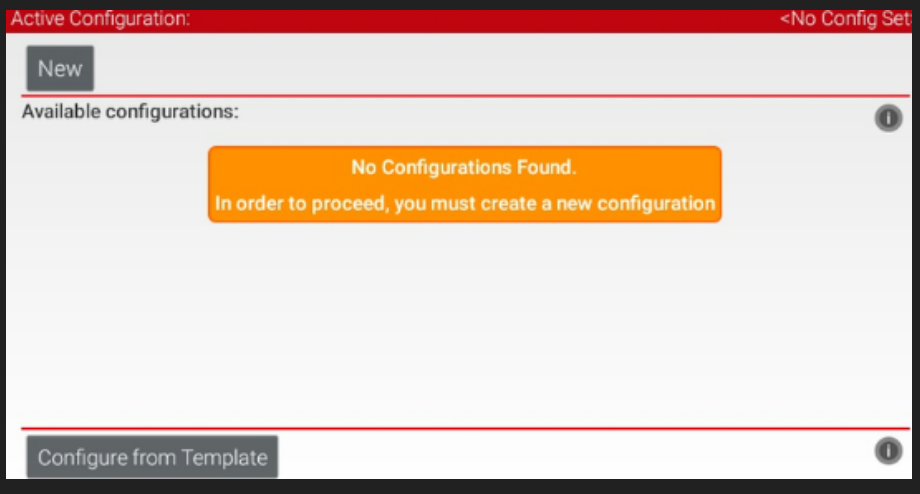
- Abra la aplicación del controlador en el teléfono del controlador del robot o en la pantalla de la estación del conductor para Control Hub.
- Haga clic en los tres puntos verticales en la esquina superior derecha de la pantalla.

Seleccionar Ajustes y luego Configurar robot

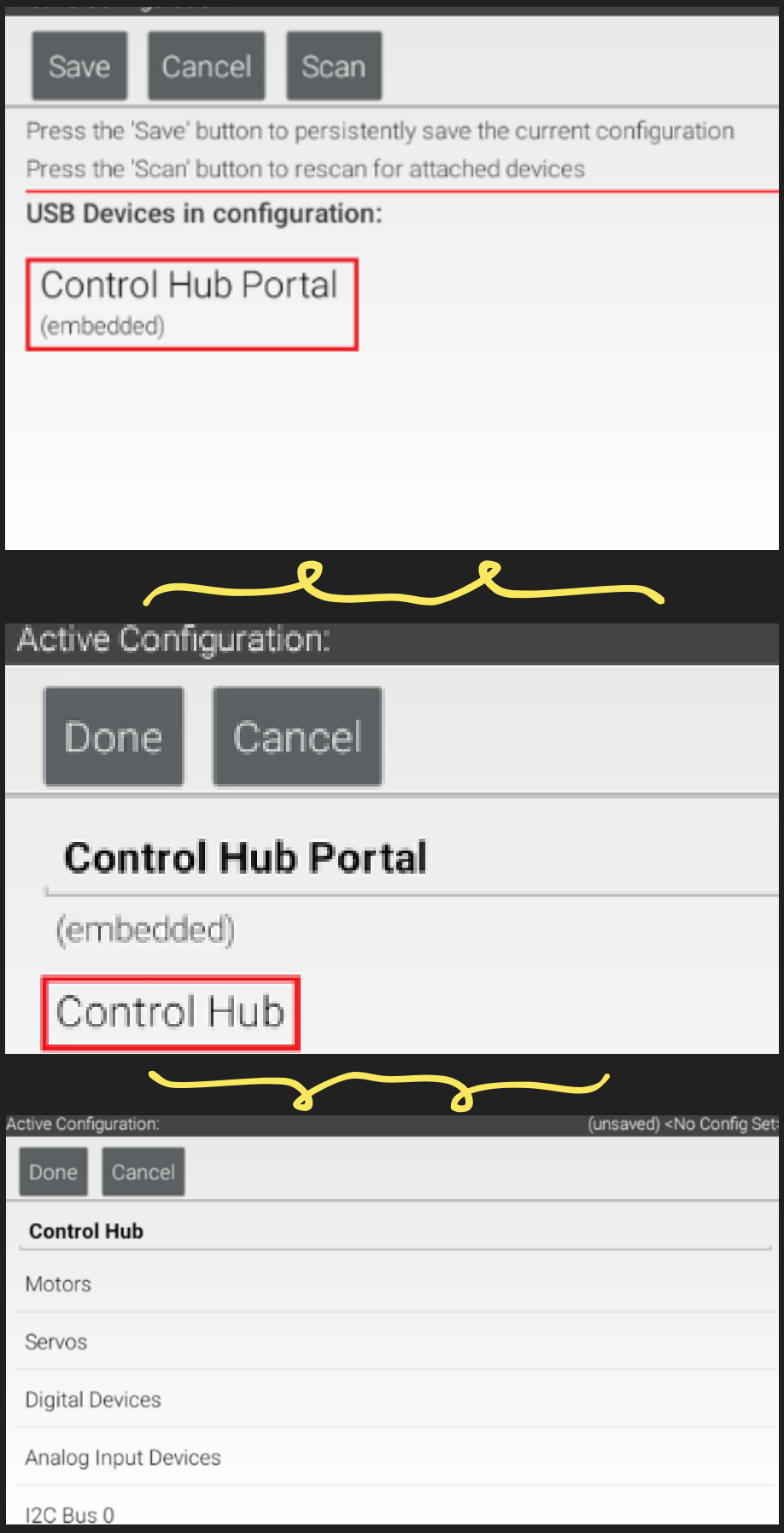


Aquí verá una lista de los archivos de configuración existentes, si los hay.

- Si no existe ningún archivo, se le pedirá que cree uno nuevo.



- Puede editar, eliminar o activar archivos existentes según sea necesario.

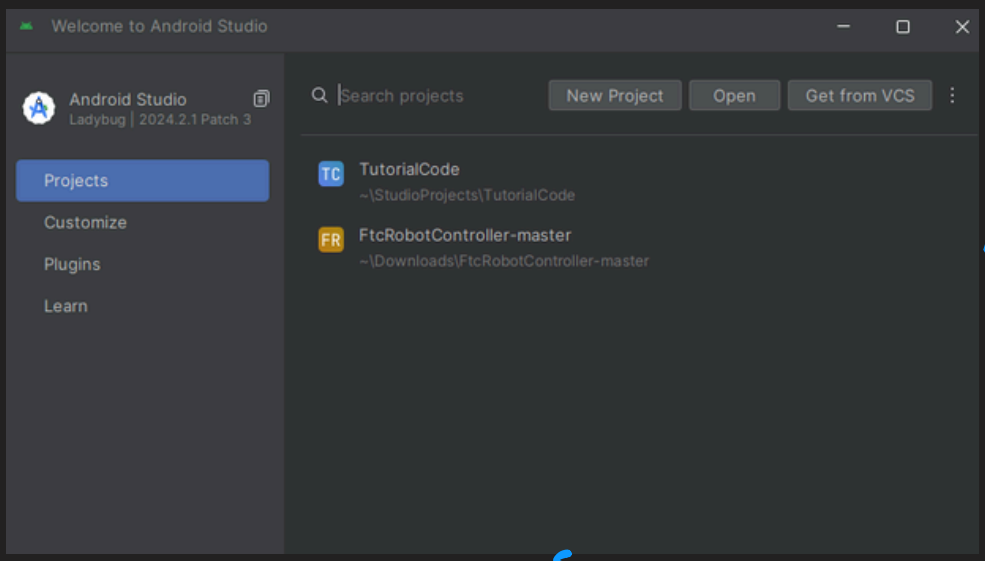


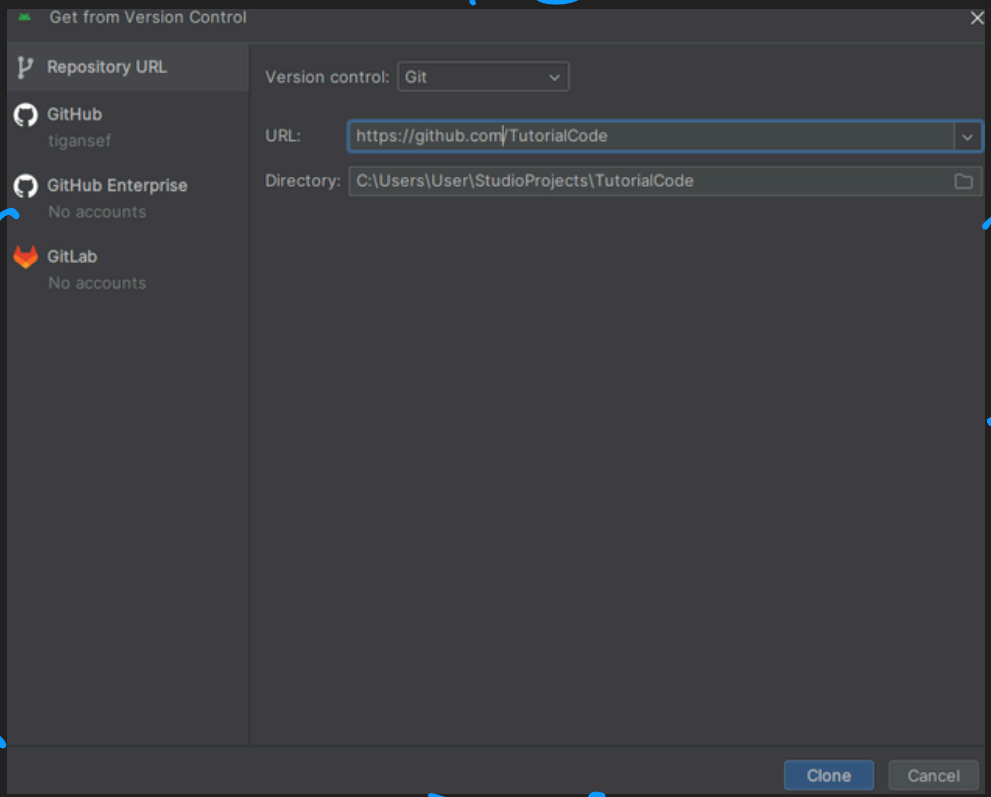
El mapeo de hardware es un paso crucial en la programación de su robot. Garantiza que su software y hardware estén conectados correctamente, sentando las bases para escribir programas que puedan controlar su robot de manera efectiva.

1.5.Guía de instalación del CONTROLADOR DEL ROBOT FTC

El controlador de robot FTC es un repositorio de GitHub que contiene el código fuente que se utiliza para crear una aplicación de Android para controlar un robot de competencia FIRST Tech Challenge. Para descargar este repositorio, debe abrir el sitio oficial de controladores de robots ftc.Allí verás la última edición.

En la esquina superior derecha de la página, verá la opción de crear una bifurcación del repositorio. Eso significa crear una copia tuya que sea editable. Para abrirlo en Android Studio, haga clic en **obtener de VCS** y pegue el enlace de su bifurcación recién creada. ¡Ahora espera a que Gradle se sincronice y estarás listo para codificar!





2.CONCEPTOS BÁSICOS DE PROGRAMACIÓN

2.1.¿Qué es un modo de operación?

Un OpMode es un modo operativo que hace que el hardware funcione y le dicta directamente al robot cómo debe comportarse. Existen múltiples tipos de OpModes, cada uno para diferentes propósitos, por ejemplo, hay OpModes que se usan para el periodo autónomo y otros para TeleOp.

En el período autónomo, es importante tener en cuenta que debido a que no hay gamepads y no hay interacción humana directa con el robot, las instrucciones deben estar preprogramadas y debes confiar en sensores.

En TeleOp basta con usar los gamepads, pero automatizar algunas cosas podría ser muy útil para los conductores en el partido. Al programar para TeleOp debes hacer que el robot se mueva usando los botones de los gamepads.

en el contexto de la FTC, se deben ampliar dos clases al crear su propio OpMode, porque contienen métodos clave que son esenciales para escribiendo codigo para hacer funcionar el robot: LinearOpMode y OpMode. Puedes ampliar cualquiera de estas clases, depende de tu preferencia; Ambas clases tienen sus métodos:

¿CÓMO CREAR UN OPMODE?

- Estos son los métodos que puede utilizar para crear su OpMode:

Métodos LinearOpMode

ejecutarModoOp() el código de este método se ejecuta una vez después de presionar INIT - aquí debería estar todo el código para su OpMode

waitForStart(): pausa el modo de operación hasta que presione INICIO en la estación del conductor

está iniciado() devuelve verdadero si se presiona el botón INICIO; de lo contrario, devuelve falso

isStopRequested() devuelve verdadero si se presiona el botón DETENER; de lo contrario, devuelve falso

inactivo(): pausa temporalmente el hilo y le da tiempo al procesador para manejar otras tareas (como actualizaciones de telemetría)

Métodos OpMode

init() se ejecuta exactamente una vez después de presionar INIT en la estación del conductor

bucle_inicial() una vez que se haya ejecutado el código en init(), el código dentro de este método se ejecutará continuamente hasta que se presione INICIO

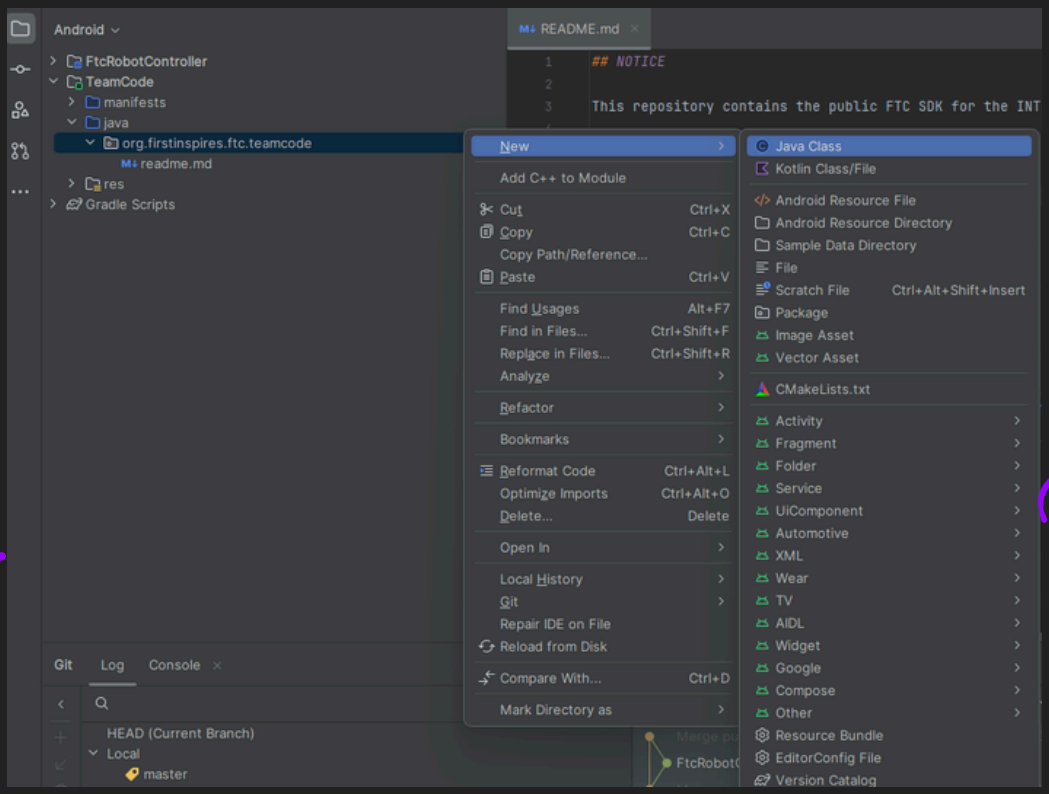
start(): el código dentro de este método se ejecutará exactamente una vez después de presionar el botón INICIO en la estación del conductor.

loop(): una vez que se haya ejecutado el código en start(), el código aquí se ejecutará continuamente hasta que se presione el botón STOP

stop(): el código dentro de este método funcionará una vez que presione DETENER

2.2.¿Qué es un modo de operación?

Para comenzar a programar un motor debes introducirlo en el mapa de hardware como se mostró anteriormente. Luego, podemos comenzar a crear un nuevo OpMode donde se programará el motor.
En primer lugar, en org.firstinspires.ftc.teamcode, cree una clase java.fig 2.3.1.



- En esta clase debes comenzar escribiendo el código de modo de operación clásico así.

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

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

no usages
@TeleOp(name="motorProgrammingTeleop")

public class motorProgrammingTutorial extends LinearOpMode {

    @Override

    public void runOpMode() throws InterruptedException
    {
        //Initialization Code is written here

        waitForStart();

        while(opModeIsActive())
        {

        }
    }
}
```

- Ahora puedes crear un objeto variable DcMotor. en el espacio de inicialización con cualquier nombre. Para que el código funcione, debe importar la clase DcMotor colocando el cursor sobre ella y presionando Alt+Shift+Enter.

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

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

no usages
@TeleOp(name="motorProgrammingTeleop")

public class motorProgrammingTutorial extends LinearOpMode {

    @Override

    public void runOpMode() throws InterruptedException
    {
        //Initialization Code is written here

        waitForStart();

        while(opModeIsActive())
        {

        }
    }
}
```

- A continuación debe especificar qué motor en el mapa de hardware corresponde a la variable recién creada. El nombre del dispositivo que ha escrito en la Configuración debe ser exactamente el mismo en el código.

```java
DcMotor tutorialMotor;
tutorialMotor = hardwareMap.get(DcMotor.class, deviceName: "tutorial_Motor");
```

- Ahora, hay algunas variables que puede configurar para afectar el funcionamiento del motor de CC.El primero de ellos es la dirección.

```java
tutorialMotor.setDirection(DcMotor.Direction.REVERSE);
tutorialMotor.setDirection(DcMotor.Direction.FORWARD);
```

- Cambiar la dirección del motor hace exactamente lo que se debería esperar: cambia la dirección. Si se aplica una potencia de 1 al motor mientras está en modo de avance, girará en una dirección. Si está al revés, una potencia de 1 lo hará girar en la otra dirección. A continuación, hay dos comportamientos de energía cero que se pueden ajustar.

```java
tutorialMotor.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
tutorialMotor.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.FLOAT);
```

- Cambiar esta variable determina cómo se comporta el motor de CC cuando se aplica una potencia de 0. BRAKE hace que el motor intente desacelerarse si está en movimiento (pero no mantendrá su posición si ya está parado). FLOAT, por otro lado, permite que el motor se deslice hasta detenerse de forma natural, dependiendo completamente de la fricción.Por último, hay cuatro modos de funcionamiento diferentes que se pueden utilizar con motores de CC:

```
tutorialMotor.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
tutorialMotor.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
tutorialMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
tutorialMotor.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
```

- Echemos un vistazo más de cerca a los diferentes modos de codificador que puede configurar para su motor. Estos modos generalmente se establecen durante la fase de inicialización de su código, lo que garantiza que los motores estén configurados y listos para funcionar durante todo el programa. Sin embargo, puede cambiar el modo durante la ejecución si un caso de uso específico lo requiere. Seleccionar el modo correcto garantiza que su robot se desempeñe de manera eficiente y confiable para sus tareas específicas.

//STOP_AND_RESET_ENCODER

Cuando se utilizan codificadores, es una buena idea comenzar con STOP_AND_RESET_ENCODER durante la inicialización. Esto restablece la posición del motor para que sepas exactamente dónde comienza. Solo asegúrese de que su robot esté en la misma posición inicial cada vez para mantener la coherencia.
También puede restablecer el codificador mientras el robot está en funcionamiento. Muchos equipos configuran un botón en el mando para hacer esto. Es útil si el robot se atasca o un motor no funciona correctamente, ya que te permite fijar rápidamente la posición del motor durante una partida.

//RUN_WITHOUT_ENCODER

En este modo, proporciona un nivel de potencia en el rango de -1 a 1, donde -1 es velocidad máxima hacia atrás, 0 es parado y 1 es velocidad máxima hacia adelante. Reducir la potencia reduce tanto el par como la velocidad. Como ejemplo, aquí hay un código que establece la potencia del motor al máximo mientras se presiona el botón A.

```
if(gamepad1.a)
    tutorialMotor.setPower(1);
```

//RUN_USING_ENCODER

En el modo RUN_USING_ENCODER, el Control Hub utiliza el codificador para gestionar activamente la velocidad del motor. En lugar de aplicar un porcentaje fijo de potencia, este modo apunta a una velocidad específica, ajustándose automáticamente a factores como la fricción o los cambios de voltaje de la batería.
Aún puede establecer un nivel de potencia en el modo RUN_USING_ENCODER, pero no se recomienda: limita la velocidad objetivo del motor y reduce la eficiencia.
Si configura una velocidad mientras está en el modo RUN_WITHOUT_ENCODER, el motor cambiará automáticamente al modo RUN_USING_ENCODER. Al elegir una velocidad, asegúrese de que sea realista para el motor, incluso a plena carga o con batería baja.
Cuando use velocidad, debe declarar el motor como DcMotorEx en el código de la siguiente manera:

```
DcMotorEx tutorialMotor;
tutorialMotor = hardwareMap.get(DcMotorEx.class, deviceName: "tutorial_Motor");
```

Este fragmento de código establece la velocidad del motor en 300 tics/segundo cuando se presiona el botón a.

```
if(gamepad1.a)
    tutorialMotor.setVelocity(300);
```

Para saber cuántos tics tiene que hacer su motor para dar una vuelta completa, debe buscar el modelo exacto de su motor. en el sitio de Gobilda o de cualquier otro proveedor.

//RUN_TO_POSITION

En el modo RUN_TO_POSITION, el Control Hub apunta a una posición específica para el motor en lugar de centrarse en la velocidad. Puede establecer una velocidad máxima, pero solo limita la rapidez con la que se mueve el motor para alcanzar el objetivo. Una vez que el motor alcance la posición, mantendrá esa posición automáticamente.

Consideraciones clave

- Si el motor no puede alcanzar la posición objetivo, seguirá intentándolo, lo que podría provocar un sobrecalentamiento.
- Para evitar problemas, asegúrese de que el camino hacia el objetivo esté despejado y evite forzar el motor en posiciones imposibles.

Consejos importantes

- Asegúrese de que su robot esté físicamente reiniciado para que coincida con este punto de partida (por ejemplo, reposicione un brazo o mecanismo).
- Para evitar que el motor mantenga su posición, establezca la potencia o la velocidad en cero o cambie a un modo de motor diferente.

Este modo es ideal para tareas en las que el posicionamiento preciso es fundamental, como mover un brazo a una altura específica o girar un mecanismo a un ángulo fijo.

- Aquí hay un ejemplo de un código donde cuando se presiona el botón a, el motor va a la posición deseada (establecida en 1000 en el código) y cuando se presiona el botón b, vuelve a la posición inicial.

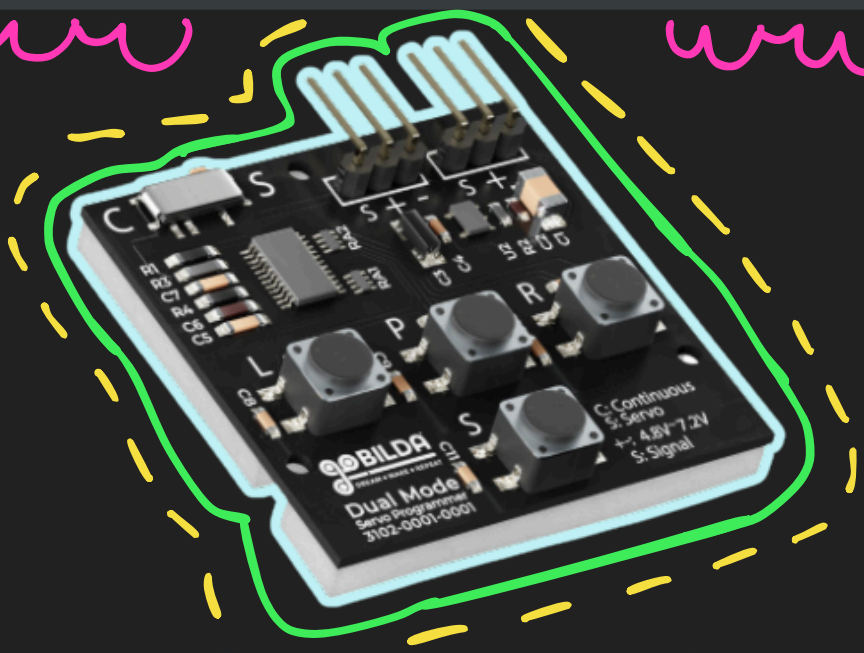
```
DcMotorEx tutorialMotor;
tutorialMotor = hardwareMap.get(DcMotorEx.class, deviceName: "tutorial_Motor");
tutorialMotor.setMode(DcMotor.RunMode.RUN_TO_POSITION);
int desiredposition=1000;
tutorialMotor.setPower(1);

waitForStart();

while(opModeIsActive())
{
    if(gamepad1.a)
    {
        tutorialMotor.setTargetPosition(desiredposition);
    }
    if(gamepad1.b)
    {
        tutorialMotor.setTargetPosition(0);
    }
}
```

2.3.tipos de servos

Hay dos tipos de servo que debes conocer, primero hay un servo normal, que gira una cantidad determinada de grados y tiene límites de posición (no puede girar infinitamente en una dirección) y el otro tipo es CRservo (servo de rotación continua).) que actúa de manera similar a un motor en el RUN_WITHOUT_ENCODER modo. Puede girar sin cesar en una dirección y, en el código, se utiliza el establecer potencia() función para hacer que se mueva.
Se puede programar un servo para que se ejecute en cualquiera de estos modos (si es un servo de modo dual). Para hacer esto, debe tener un programador de servo que se puede encontrar en el sitio de GoBilda. Tiene este aspecto.



//Configuración del servo en modo servo (predeterminado)

1. Conecte el servo y la batería al servoprogramador.
2. Desliza el interruptor de modo a "S".
3. Mantenga presionado el tecla "P" durante unos 5 segundos.
4. Cuando todos los LED parpadean, la programación se completa y el servo ahora está en **Modo servo**.

//Configuración del servo en modo de rotación continua

1. Conecte el servo y la batería al servoprogramador.
2. Presione el tecla "S" para ingresar al modo de barrido (el LED "S" parpadeará rápidamente).
3. Presione el **tecla "S"** nuevamente para ingresar al modo de prueba de 3 puntos (el LED "S" parpadeará más lento).
   - **Izquierda (tecla L):**
     - En modo Servo: Mueve el servo a la posición "Izquierda".
     - En modo de rotación continua: gira el servo a máxima velocidad hacia la izquierda.
   - **Medio (tecla P):**
     - En modo Servo: Mueve el servo a la posición "Media".
     - En modo de rotación continua: detiene el servo.
   - **Derecha (tecla R):**
     - En modo Servo: Mueve el servo a la posición "Derecha".
     - En modo de rotación continua: gira el servo a máxima velocidad hacia la derecha.

4. Presione el **tecla** "S" por tercera vez para salir del modo de prueba de 3 puntos. Todos los LED se apagarán.

Este proceso le permite configurar y verificar rápidamente la funcionalidad de su servo en ambos modos.

2.4 Programando un servo

Debido a que hay dos tipos diferentes de servos: simples y continuos, existen diferentes métodos para hacerlos moverse, así que, por un servo simple simplemente pasarías la posición deseada (un número entre 0 y 1) a un método llamado setPosition(). También puedes usar getPosition() especialmente al depurar, ya que te da la posición exacta donde se encuentra el servo en ese momento. Puedes ver un ejemplo que he hecho en esta imagen.

```java
package org.firstinspires.ftc.teamcode.drive.writtenCode;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.Servo;

public class ServoOpMode extends LinearOpMode {  no usages  new *
    @Override  new *
    public void runOpMode() throws InterruptedException {
        Servo myServo= hardwareMap.get(Servo.class, deviceName: "SERVO_CONFIG_NAME");
        waitForStart();
        while (opModeIsActive()){
            if(gamepad1.a) {
                myServo.setPosition(0.5); // position in range [0,1]
            }
            telemetry.addData( caption: "Servo position", myServo.getPosition());
            telemetry.update();
        }
    }
}
```

- Aquí hago que el servo vaya hasta la mitad cuando se presiona el botón a y en telemetría obtengo la posición exacta donde está el servo.

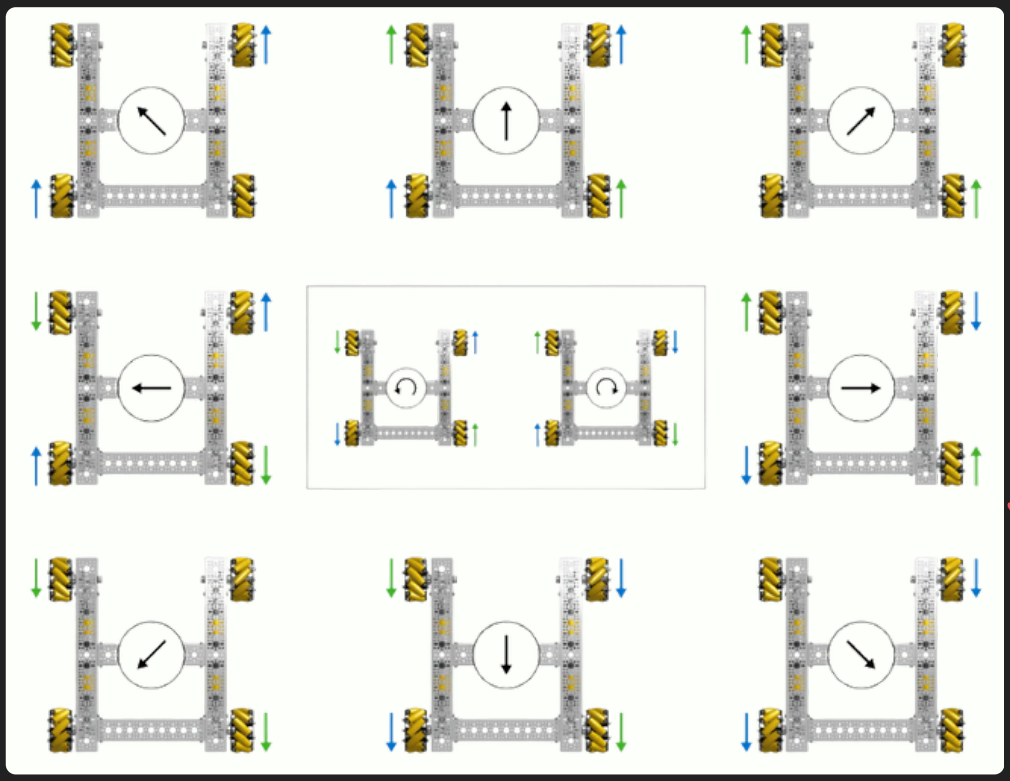
Por otro lado, al programar un servo de rotación continua (CR Servo), no dependerás de pasar la posición, sino de darle potencia pasando a setPower() un número entre -1 y 1. Cuando le das al servo potencia -1, retrocede, y cuando le das potencia 1 sigue adelante. Si le das potencia 0 el servo simplemente se detiene. Puedes ver cómo le di energía al servo a continuación.

```java
1  package org.firstinspires.ftc.teamcode.drive.writtenCode;
2
3  import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
4  import com.qualcomm.robotcore.hardware.CRServo;
5  import com.qualcomm.robotcore.hardware.DcMotorSimple;
6
7  public class ContinuousServoOpMode extends LinearOpMode {  no usages  new *
8      @Override  new *
9      public void runOpMode() throws InterruptedException {
10         CRServo myContinuousServo= hardwareMap.get(CRServo.class, deviceName: "CONT_SERVO_CONFIG_NAME");
11         //if it's necessary you can reverse the servo's direction like in line 12
12         myContinuousServo.setDirection(DcMotorSimple.Direction.REVERSE);
13         waitForStart();
14         while(opModeIsActive()) {
15             if (gamepad1.a) {
16                 myContinuousServo.setPower(1); // power must be in range [-1,1]
17             }
18         }
19     }
20 }
```

También puedes invertir la dirección en la que va el servo; no importa si es continuo o simple. Puedes ver cómo puedes hacer esto en la línea 12 de la imagen.

2.5 Programando un chasis MECANUM

Lo más importante sobre el mecano El chasis es que sus 4 ruedas tienen rodillos en un ángulo de 45 grados por lo que esto influye en el movimiento del robot, por cómo funcionan sus fuerzas.



Aquí puedes ver cómo se comporta cada una de las ruedas al circular en diferentes direcciones. No es necesario codificar el movimiento en cada dirección, porque hay una manera mucho mejor de codificar el chasis

- Lo primero que tienes que declarar, inicializar, y si quieres, overclock los motores del chasis, como en la imagen de abajo. Esto se hace porque por defecto los motores sólo funcionan al 0,8 de su potencia máxima en 1.

```
DcMotor rightFront = hardwareMap.get(DcMotor.class,"rightFront");
DcMotor leftFront = hardwareMap.get(DcMotor.class,"leftFront");
DcMotor rightBack = hardwareMap.get(DcMotor.class,"rightBack");
DcMotor leftBack = hardwareMap.get(DcMotor.class,"leftBack");
MotorConfigurationType mct1, mct2, mct3, mct4;
mct1 = rightBack.getMotorType().clone();
mct1.setAchieveableMaxRPMFraction(1.0);
rightBack.setMotorType(mct1);

mct2 = rightFront.getMotorType().clone();
mct2.setAchieveableMaxRPMFraction(1.0);
rightFront.setMotorType(mct2);

mct3 = leftFront.getMotorType().clone();
mct3.setAchieveableMaxRPMFraction(1.0);
leftFront.setMotorType(mct3);

mct4 = leftBack.getMotorType().clone();
mct4.setAchieveableMaxRPMFraction(1.0);
leftBack.setMotorType(mct4);
```

- Los nombres entre comillas deben coincidir con los nombres de los motores de CONFIG.

- Debido a la posición de los motores, debes invertir dos de tus motores. Este paso es muy importante porque si lo omite, los motores girarán en la dirección incorrecta. Esto depende de la configuración de su motor y debe asegurarse de que todos funcionen en la dirección correcta.

```
leftFront.setDirection(DcMotor.Direction.REVERSE);
leftBack.setDirection(DcMotor.Direction.REVERSE);
```

- Debe crear métodos para configurar el modo de ejecución deseado (use RUN_WITHOUT_ENCODER) y el comportamiento de energía cero (use BRAKE) y hacer que el robot funcione implementando una unidad centrada en el robot, como en la imagen a continuación.. También debes crear y copiar los dos gamepads que utilices. Se puede utilizar FLOAT en lugar de BRAKE para que el robot no mantenga su posición al frenar.

```
public void robotCentricDrive(DcMotor leftFront , DcMotor leftBack,
                              DcMotor rightFront ,DcMotor rightBack,
                              double leftTrigger, double rightTrigger)
{
    /// O sa va intrebati cum putem accesa gamepad1 si gamepad2 ?
    /// Probabil sunt variabile globale , n-ar trebui sa va faceti multe griji

    double y = -gamepad1.left_stick_y; // Remember, Y stick value is reversed
    double x = (-gamepad1.left_trigger + gamepad1.right_trigger)* 1.05; // 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 leftFrontPower = (y + x + rx) / denominator;
    double leftBackPower = (y - x + rx) / denominator;
    double rightFrontPower = (y - x - rx) / denominator;
    double rightBackPower = (y + x - rx) / denominator;


    leftFront.setPower(leftFrontPower);
    leftBack.setPower(leftBackPower);
    rightFront.setPower(rightFrontPower);
    rightBack.setPower(rightBackPower);
}
```

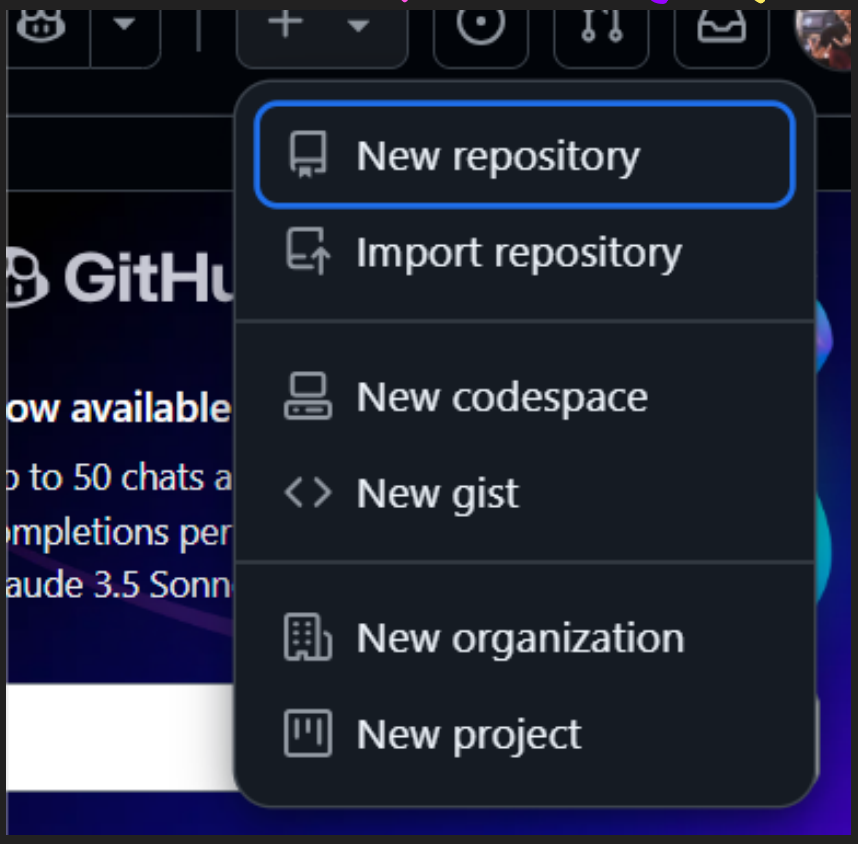
2.6 GitHub

Como se escribió anteriormente, los programadores tienen que trabajar juntos y la plataforma principal que utilizan los codificadores para compartir su código es GitHub. Debido a que los equipos de la FTC se ocupan de muchas iteraciones, tendrías que realizar un seguimiento de los cambios que realizas y ahí es donde entra en juego Git (un sistema de control de versiones). Básicamente, cada cambio que realizas en tu repositorio se rastrea y se almacena en confirmaciones. . Si algo sale mal, aparece un error, los equipos podrían simplemente retroceder a la versión anterior del código y corregirlo, por eso es tan útil y ampliamente utilizado. El sistema rastrea automáticamente los cambios y se asegura de que todo esté actualizado.

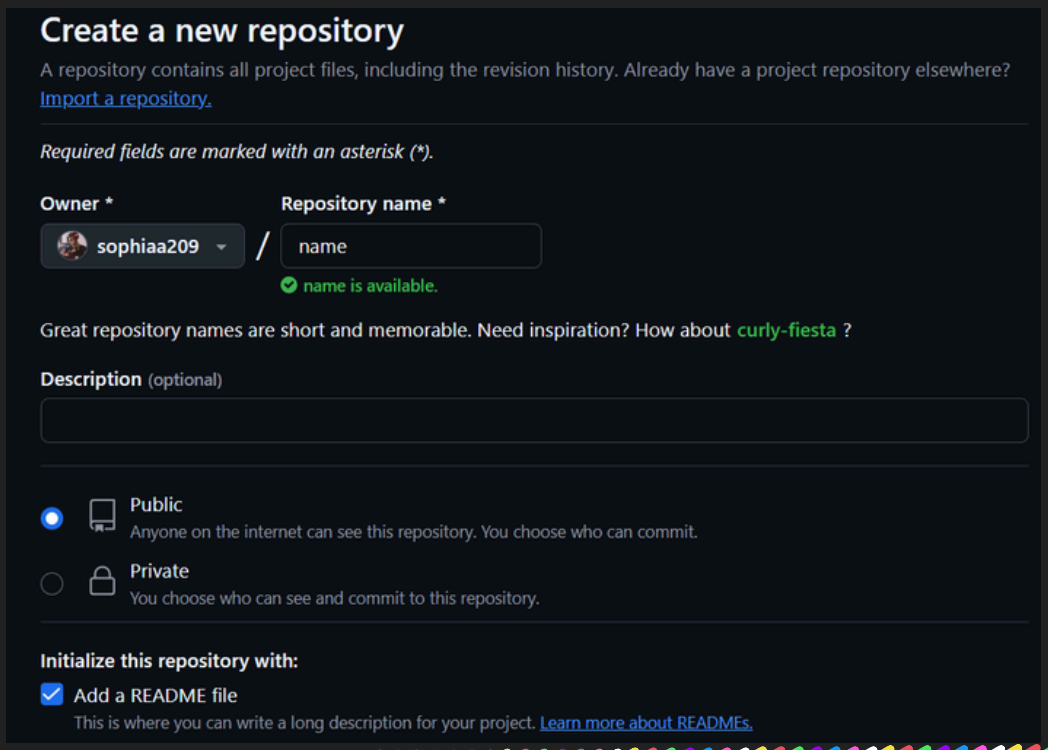
Como hay varios programadores, todos pueden acceder a un repositorio centralizado al que todos tienen acceso. Así es como funciona la colaboración. El proceso de usar GitHub: crear un repositorio y compartirlo es muy intuitivo y me sumergiré en eso. Puedes simplemente crear una cuenta y crear tu propio repositorio.

- Para crear un repositorio, haga clic en el ícono más en la barra superior,

- Haga clic en "Nuevo repositorio".

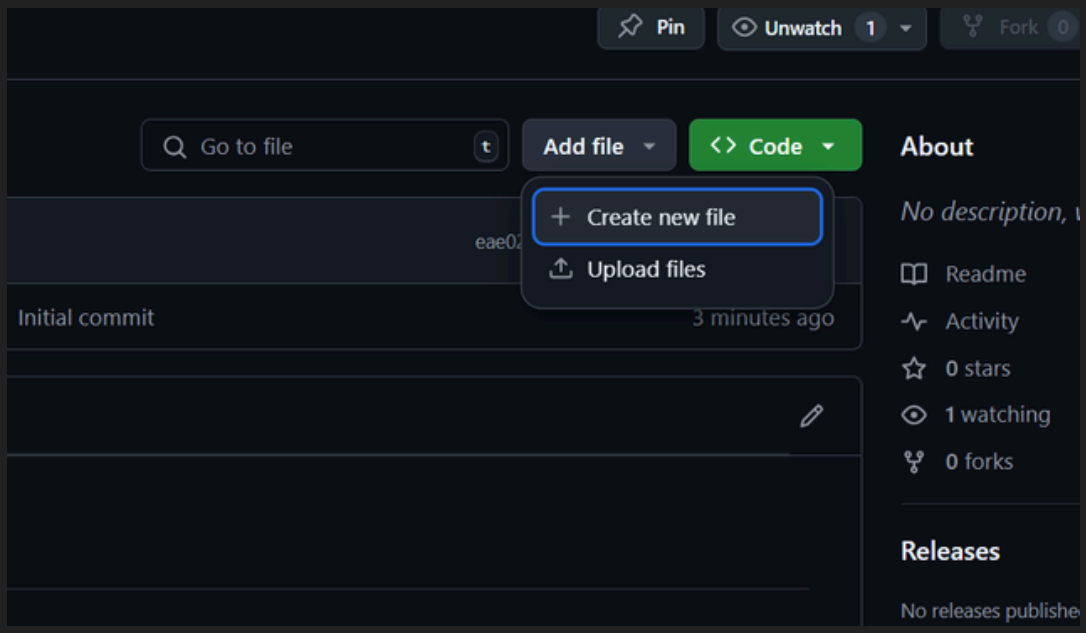


- Dale un nombre a tu repositorio, hazlo público si quieres que alguien más lo vea y agrega un archivo Léame.

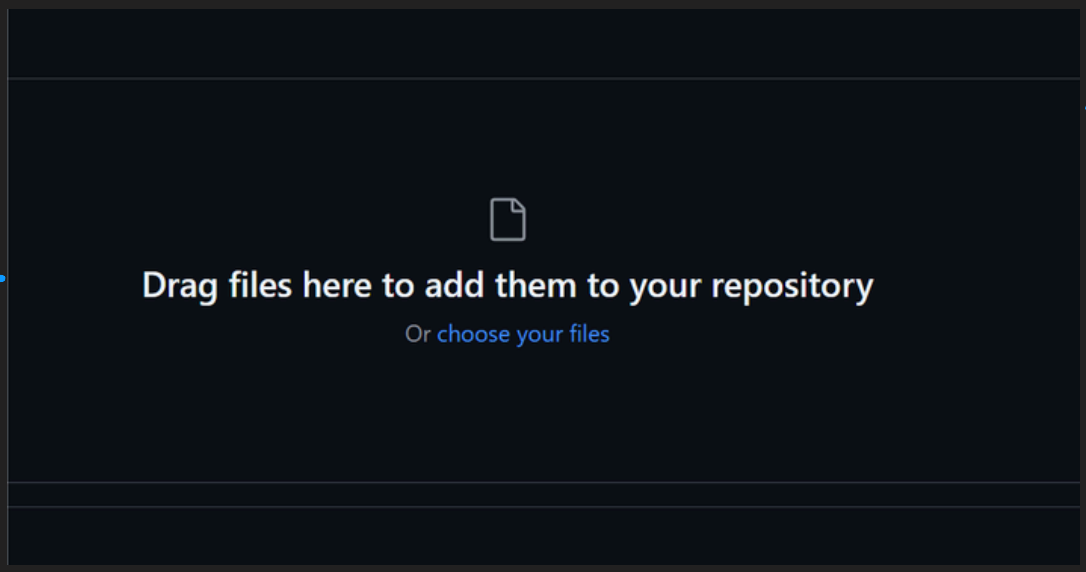


- y en la esquina inferior derecha haga clic en "Crear repositorio". Puede editar el archivo Léame que aparece automáticamente.

Para agregar el archivo que deseas haz clic en “Agregar archivo”



- y cárgalo en tu proyecto arrastrándolo aquí.



Una vez que haya terminado de cargar sus archivos, puede compartir su repositorio copiando su enlace y enviándolo.

3. FORMAS DE PROGRAMAR UN ROBOT SIMPLE

3.1 Introducción a las máquinas de estados finitos

Las máquinas de estados finitos (FSM) son muy útiles para programar acciones más complejas. Son especialmente útiles si tienes varias tareas ejecutándose al mismo tiempo, ya que permiten que esas tareas interactúen y dependan entre sí de una manera más flexible y no lineal.

//¿QUÉ ES UNA MÁQUINA DE ESTADO FINITO?

Una máquina de estados finitos (FSM) es exactamente lo que parece: es un sistema con un número limitado de estados. En cualquier momento dado, se encuentra en un estado específico y puede cambiar (o hacer una transición) a un estado diferente cuando algo desencadena ese cambio.
En los siguientes capítulos, donde verá cómo nuestro equipo codifica un robot, aprenderá a utilizar este concepto de programación.

3.2 EJEMPLO DE CÓDIGO

3.2.1 RobotMapa clase

Básicamente, RobotMap es una clase que contiene todo el hardware que desea codificar, motores, sensores, etc. (excepto los motores del chasis que se declaran e inicializan en la clase TeleOpCode). Es como un contenedor para todo el hardware que desea controlar o leer datos durante el funcionamiento de su robot. RobotMap es como una capa de abstracción entre los componentes de hardware y el resto del programa, lo que le permite acceder fácilmente y configurar múltiples dispositivos en la configuración de hardware del robot. La lógica principal de su robot permanece organizada de esta manera.

Tienes que hacer un constructor en esta clase (que tome como parámetro un objeto HardwareMap). Dentro del constructor hay que inicializar los componentes de hardware que utilizará el robot durante sus operaciones. El constructor establece las conexiones necesarias entre su software y hardware.

Cuando crea un objeto, una instancia de esta clase, utiliza este constructor y le pasa el objeto HardwareMap. Eso asegura que el hardware esté inicializado.
Tener una clase RobotMap centraliza la configuración del hardware, hace que el código sea más limpio y fácil de mantener.

Veamos un ejemplo concreto. En la imagen siguiente puedes ver cómo se declaran los componentes de hardware:

```
public class RobotMap {  16 usages  new *

    public Servo clawRotate;  2 usages
    public Servo clawLeft;  2 usages
    public Servo clawRight;  2 usages
    public Servo linkage1;  2 usages
    public Servo linkage2;  2 usages
    public Servo clawPosition;  2 usages
    public Servo fourbar;  2 usages
    public Servo pivot1;  2 usages
    public Servo pivot2;  2 usages


    public DcMotorEx lift1;  7 usages
    public DcMotorEx lift2;  6 usages
    public MotorConfigurationType mctlift1, mctlift2;  no usages
```

- Luego, después de declararlos, necesitamos crear el constructor de esta clase e inicializar, como se dijo antes, los componentes de hardware de esta manera:

```
public RobotMap(HardwareMap Init)  1 usage  new *
{
    lift1=Init.get(DcMotorEx.class, deviceName: "lift1");
    lift1.setDirection(DcMotor.Direction.FORWARD);
    lift1.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
    lift1.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    lift1.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

    lift2=Init.get(DcMotorEx.class, deviceName: "lift1");
    li[illegible].setDirection(DcMotor.Direction.FORWARD);
    lift2.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
    lift2.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);
    lift2.setMode(DcMotor.RunMode.RUN_USING_ENCODER);

    linkage1=Init.get(Servo.class,  deviceName: "linkage1");
    linkage2=Init.get(Servo.class,  deviceName: "linkage2");
    clawRotate=Init.get(Servo.class,  deviceName: "clawRotate");
    clawLeft=Init.get(Servo.class,  deviceName: "clawLeft");
    clawRight=Init.get(Servo.class,  deviceName: "clawRight");
    clawPosition=Init.get(Servo.class,  deviceName: "clawPosition");
    fourbar=Init.get(Servo.class,  deviceName: "fourbar");
    pivot1=Init.get(Servo.class,  deviceName: "pivot1");
    pivot2=Init.get(Servo.class,  deviceName: "pivot2");

}
```

3.2.2 Muestras de FSM

```
switch(currentStatus)
{
    case OPEN:
    {
        this.clawLeft.setPosition(open_position);
        this.clawRight.setPosition(closed_position);

        break;
    }

    case CLOSED:
    {
        this.clawRight.setPosition(open_position);
        this.clawLeft.setPosition(closed_position);
        break;
    }

}
```

- Como puede ver en este primer ejemplo, hay dos casos dentro de este FSM: ABIERTO y CERRADO. Estos dos casos determinan la posición de la garra. Si el estado es ABIERTO, la garra está abierta; de lo contrario, la garra está cerrada.

```
switch(currentStatus)
{

    case INIT:
    {
        this.fourbar.setPosition(init_position);
        break;
    }

    case HIGH:
    {
        this.fourbar.setPosition(high_position);
        break;
    }

}
```

- Éste es otro ejemplo en el que la posición de un cuatro barras se maneja. Aquí también hay dos estados: INIT y HIGH. Si estamos en el caso INIT, la posición de la barra delantera sería la de init y si estamos en la posición HIGH, la cuatro barras ascensores.

```
switch (currentStatus) {

    case INIT: {
        this.lift1.setPower(0.5);
        this.lift2.setPower(0.5);
        currentPosition = init_position;
        break;
    }

    case SCORE: {
        currentPosition=score_position;
        break;
    }

}
```

- En este ejemplo puedes ver 2 estados de un mecanismo de elevación: INIT y SCORE. Nuevamente, por más intuitivo que parezca, en el estado INIT el elevador está en su posición inicial y en el estado SCORE está en su posición de puntuación.

```
switch(currentStatus)
{

    case INIT:
    {
        this.clawPosition.setPosition(init_position);
        break;
    }

    case COLLECT:
    {
        this.clawPosition.setPosition(collect_position);
        break;
    }

    case PLACESPECIMEN:
    {
        this.clawPosition.setPosition(specimen_position);
    }

}
```

- En este ejemplo hay 3 estados: INIT, COLLECT y PLACESPECIMEN. Una vez más, los nombres son sugerentes. En el estado INIT la posición de la garra es la inicial, en el estado COLLECT la garra está en su posición para recolectar y en el estado PLACESPECIMEN la posición de la garra es para colocar una muestra.

3.2.3 Teleoperado

TeleOp es un OpMode (modo operativo) que se basa en entradas de gamepads para controlar el robot durante su funcionamiento, a diferencia del código automático que se basa en entradas de sensores e instrucciones preprogramadas para realizar tareas específicas sin control humano en tiempo real.

Si bien Auto está diseñado para funcionar de forma autónoma en función de secuencias preprogramadas, TeleOp brinda a los conductores control total sobre el robot. Les permite conducirlo e interactuar con múltiples mecanismos presionando botones y movimientos del joystick en el gamepad. Para comprender mejor qué es un OpMode y cómo crear uno, consulte la sección específica de esta guía.
En TeleOp, el objetivo principal es asignar los botones del gamepad y los joysticks a acciones específicas en el robot. Como ejemplo, podría pensar en asignar el botón a para cerrar y abrir una garra. Hay varios botones en el gamepad que se pueden usar.

- Necesitamos crear 3 métodos antes de usar runOpMode(), eche un vistazo a las siguientes imágenes de nuestro código teleOp para nuestro robot.

```
public void setMotorRunningMode(DcMotor leftFront, DcMotor leftBack, DcMotor rightFront,  1 usage  new *
                                 DcMotor rightBack, DcMotor.RunMode runningMode) {
    leftFront.setMode(runningMode);
    rightFront.setMode(runningMode);
    leftBack.setMode(runningMode);
    rightBack.setMode(runningMode);
}
```

```
public void setMotorZeroPowerBehaviour(DcMotor leftFront, DcMotor leftBack, DcMotor rightFront,  1 usage  new *
                                       DcMotor rightBack, DcMotor.ZeroPowerBehavior zeroPowerBehavior) {
    leftFront.setZeroPowerBehavior(zeroPowerBehavior);
    rightFront.setZeroPowerBehavior(zeroPowerBehavior);
    leftBack.setZeroPowerBehavior(zeroPowerBehavior);
    rightBack.setZeroPowerBehavior(zeroPowerBehavior);
}
```

```
public void robotCentricDrive(DcMotor leftFront, DcMotor leftBack,  1 usage  new *
                              DcMotor rightFront, DcMotor rightBack,
                              double leftTrigger, double rightTrigger) {

    double y = -gamepad1.left_stick_y; // Remember, Y stick value is reversed
    double x = (-gamepad1.left_trigger + gamepad1.right_trigger) * 1.05; // Counteract imperfect strafing
    double rx = gamepad1.right_stick_x;

    double denominator = Math.max(Math.abs(y) + Math.abs(x) + Math.abs(rx), 1);
    double leftFrontPower = (y + x + rx) / denominator;
    double leftBackPower = (y - x + rx) / denominator;
    double rightFrontPower = (y - x - rx) / denominator;
    double rightBackPower = (y + x - rx) / denominator;


    leftFront.setPower(leftFrontPower);
    leftBack.setPower(leftBackPower);
    rightFront.setPower(rightFrontPower);
    rightBack.setPower(rightBackPower);
}
```

- Como puede ver en la imagen a continuación, tenemos que crear una instancia de la clase RobotMap y cada uno de los controladores que codificamos para cada mecanismo y luego llamar al método update().

```
RobotMap robot= new RobotMap(hardwareMap);

ClawRotateController clawRotateController = new ClawRotateController(robot);
ClawPositionController clawPositionController = new ClawPositionController(robot);
FourbarController fourbarController = new FourbarController(robot);
LinkageController linkageController = new LinkageController(robot);
LiftController liftController = new LiftController(robot);
PivotController pivotController = new PivotController(robot);
ClawController clawController = new ClawController(robot);


clawRotateController.update();
clawPositionController.update();
fourbarController.update();
liftController.update(liftTargetPosition);
clawController.update();
pivotController.update();
linkageController.update();
```

- Lo siguiente que tenemos que hacer es inicializar los motores del chasis y hacer overclock para llevarlos a su máximo potencial.

```
//initializing chassis motors
DcMotor rightFront = hardwareMap.get(DcMotor.class, deviceName: "rightFront");
DcMotor leftFront = hardwareMap.get(DcMotor.class, deviceName: "leftFront");
DcMotor rightBack = hardwareMap.get(DcMotor.class, deviceName: "rightBack");
DcMotor leftBack = hardwareMap.get(DcMotor.class, deviceName: "leftBack");


//overclocking chassis motors
MotorConfigurationType mct1, mct2, mct3, mct4;
mct1 = rightBack.getMotorType().clone();
mct1.setAchieveableMaxRPMFraction(1.0);
rightBack.setMotorType(mct1);

mct2 = rightFront.getMotorType().clone();
mct2.setAchieveableMaxRPMFraction(1.0);
rightFront.setMotorType(mct2);

mct3 = leftFront.getMotorType().clone();
mct3.setAchieveableMaxRPMFraction(1.0);
leftFront.setMotorType(mct3);

mct4 = leftBack.getMotorType().clone();
mct4.setAchieveableMaxRPMFraction(1.0);
leftBack.setMotorType(mct4);
```

- Luego tenemos que llamar a los métodos que creamos anteriormente en este capítulo.

```
setMotorRunningMode(leftFront,leftBack,rightFront,rightBack,
        DcMotor.RunMode.RUN_WITHOUT_ENCODER);

leftFront.setDirection(DcMotor.Direction.REVERSE);
leftBack.setDirection(DcMotor.Direction.REVERSE);

setMotorZeroPowerBehaviour(leftFront,leftBack,rightFront,rightBack,
        DcMotor.ZeroPowerBehavior.BRAKE);
```

- Tenemos que crear los objetos de los gamepads:

```
// see why you need previousGamepad1 & 2 on gm0
Gamepad currentGamepad1 = new Gamepad();
Gamepad currentGamepad2 = new Gamepad();

Gamepad previousGamepad1 = new Gamepad();
Gamepad previousGamepad2 = new Gamepad();
```

```
waitForStart();

GlobalTimer.reset();

while(opModeIsActive())
{
    if(isStopRequested()) return;

    int liftCurrentPosition = robot.lift1.getCurrentPosition();

    robotCentricDrive(leftFront,leftBack,rightFront,rightBack,gamepad1.left_trigger,gamepad1.right_trigger);

    previousGamepad1.copy(currentGamepad1);
    previousGamepad2.copy(currentGamepad2);

    currentGamepad1.copy(gamepad1);
    currentGamepad2.copy(gamepad2);
```

- en while(opModeIsActive()) colocas la asignación real entre los botones y los movimientos reales de los mecanismos. Echa un vistazo a este ejemplo. Aquí, si se presiona el botón y la garra se abre y si no, la garra se cierra.

```
if(currentGamepad2.y && !previousGamepad2.y)
{
    if(clawController.currentStatus== ClawController.clawStatus.CLOSED){
        clawController.currentStatus= ClawController.clawStatus.OPEN;
    } else {
        clawController.currentStatus= ClawController.clawStatus.CLOSED;
    }
}
```

- Luego de codificar los botones debes llamar nuevamente a la función de actualización para cada uno de los controladores:

```
clawRotateController.update();
clawPositionController.update();
fourbarController.update();
liftController.update(liftTargetPosition);
clawController.update();
pivotController.update();
linkageController.update();
```

3.3 ROADRUNNER

//¿QUÉ ES ROADRUNNER?

Road Runner es una biblioteca diseñada para usarse en la planificación del movimiento en el período autónomo del partido, manteniendo el control de la velocidad y la aceleración para que los robots tengan capacidades de seguimiento de ruta más precisas. La mayoría de los equipos prefieren esta biblioteca a alternativas como Pure Pursuit y Pedro Pathing, debido a su facilidad de uso, coherencia y soporte sólido para funciones avanzadas de planificación de rutas. Por eso, al iniciar el movimiento autónomo por caminos, Road Runner es una buena opción.

Sin embargo, la desventaja es que RoadRunner ya no se mantiene activamente, por lo que cuando los equipos comienzan a aprender a usarlo deben tener esto en cuenta, porque pueden enfrentar limitaciones para adaptarse a necesidades futuras.

PROCESO DE INSTALACIÓN

Después de haber instalado el inicio rápido, tendrás que agregar algunos fragmentos en Gradle y descargar algo, así que comencemos a ello.

Hacer clic en construir.dependencias.gradle en la raíz de su proyecto.
Al final del bloque de código de los repositorios, agregue el siguiente fragmento experto `{url = 'https://maven.brott.dev/'}`

Luego, al final del bloque de dependencias, agregue esto implementación `'com.acmerobotics.dashboard:dashboard:0.4.15'`

Lo siguiente que tendrás que abrir es el gradle de TeamCode (Código de equipo/build.gradle) y agregue el siguiente código al final del bloque de dependencias

```
implementación 'org.apache.commons:commons-math3:3.6.1'
implementación 'com.fasterxml.jackson.core:jackson-databind:2.12.7'
implementación 'com.acmerobotics.roadrunner:core:0.5.6'
```

Tienes que descargar este inicio rápido de Road Runner haciendo clic en el botón verde "Código" y luego en "Descargar ZIP". acmerobotics/road-runner-quickstart en inicio rápido1

Ir al Código de equipo carpeta en el proyecto y mueva la unidad, secuencia de trayectoria y carpetas util en java/org/firstinspires/ftc/teamcode.

TRAYECTORIA

Ten en cuenta que esta es una explicación muy sencilla y antes de comenzar a hacer trayectorias, debes seguir y leer. aprenderroadrunner.com. Esta es una explicación sobre cómo instalar una versión obsoleta pero aún funcional de RoadRunner, 0.5.6. Quizás quieras investigar cómo usar 1.x en tu propio proyecto.

Ahora que tienes instalado RoadRunner puedes empezar a probar varias secuencias de trayectoria con Meep Meep. ¡Echemos un vistazo a algunos ejemplos!

Es muy importante tener en cuenta que algunos de los métodos aceptan como parámetros plantea y otros aceptan vectores. Ahora quizás te preguntes cuál es la diferencia entre un vector y una pose y la respuesta es simple: un vector contiene solo las dos coordenadas xey y una pose también contiene un encabezado.

Además, debes tener cuidado al crear trayectorias para no realizar giros bruscos, porque debes tener en cuenta las leyes de la física y el hecho de que el impulso juega un papel muy importante en los movimientos de tu robot.

Es importante priorizar la realización de splines, porque aseguran un movimiento continuo y reducen el tiempo de los movimientos y esto es muy importante porque debes asegurarte de hacer lo mejor que puedas en 30 segundos, ya que es lo que dura el auto.

Puedes ver en la imagen a continuación cómo debería verse la estructura de tu código. Esto está en la clase MeepMeepTesting. Cuando trabajas con coordenadas, puedes pasar el cursor y verás en la esquina inferior izquierda cómo cambian activamente y las tomas desde allí y las usas en tu código.

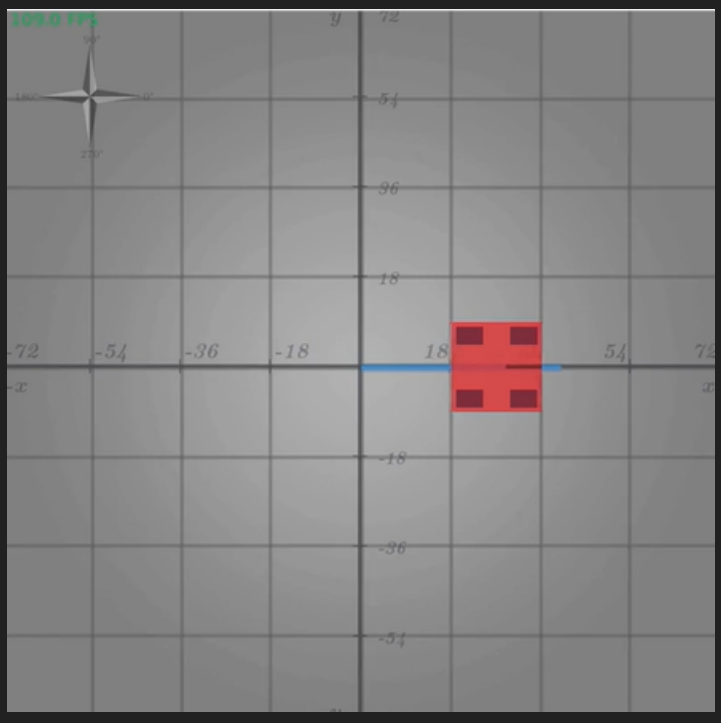
```
public class MeepMeepTesting {  catbox *
    public static void main(String[] args) {  catbox *
        MeepMeep meepMeep = new MeepMeep( windowSize: 800);

        RoadRunnerBotEntity myBot = new DefaultBotBuilder(meepMeep)
                // Set bot constraints: maxVel, maxAccel, maxAngVel, maxAngAccel, track width
                .setConstraints( maxVel: 60,  maxAccel: 60, Math.toRadians(180), Math.toRadians(180),  trackWidth: 15)
                .followTrajectorySequence(drive -> drive.trajectorySequenceBuilder(new Pose2d( x: 0,  y: 0,  heading: 0))
                        //put the methods you want here
                        .forward( distance: 30)
                        .build());


        meepMeep.setBackground(MeepMeep.Background.FIELD_INTOTHEDEEP_JUICE_DARK)
                .setDarkMode(true)
                .setBackgroundAlpha(0.95f)
                .addEntity(myBot)
                .start();
    }
}
```

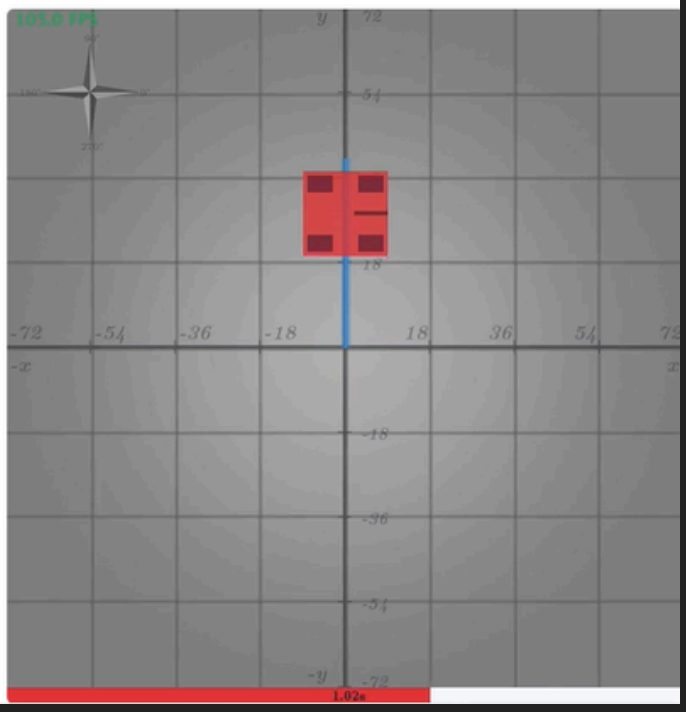
- Estos son los movimientos que puedes generar (puedes ver los métodos que debes usar en la esquina superior izquierda de cada una de las imágenes y los parámetros que debes pasar):

ADELANTE



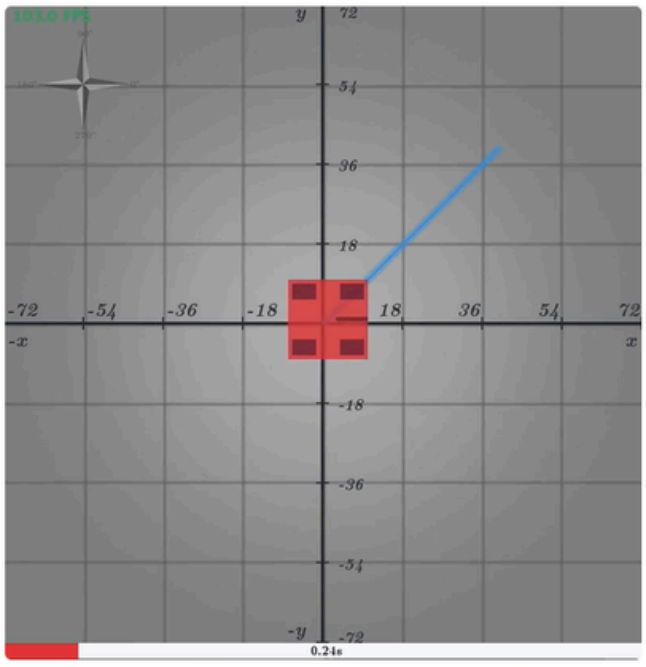
AVANZAR A LA IZQUIERDA

```
.strafeLeft(distance: Double)
```



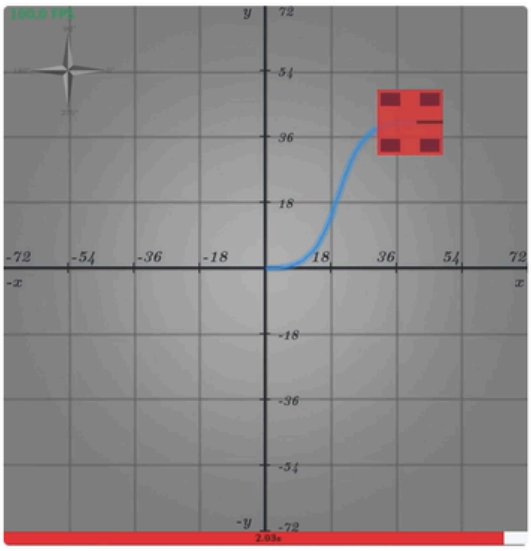
SPLINE A

```
.strafeTo(endPosition: Vector2d)
```



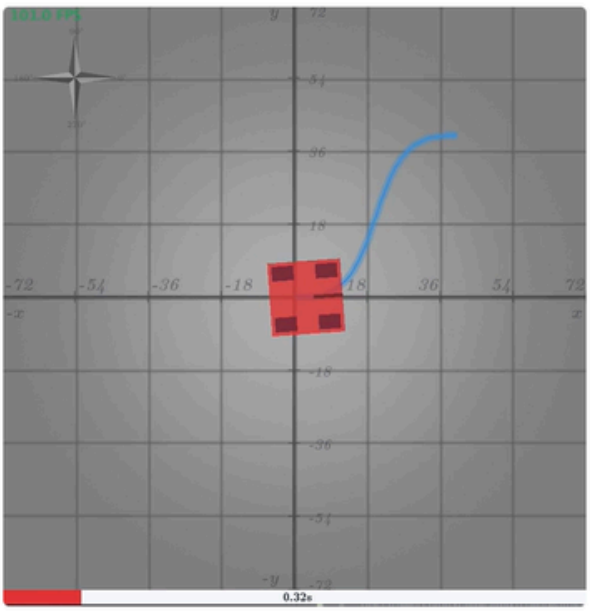
SPLINE A RUMBO CONSTANTE

```
.splineToConstantHeading(endPosition: Vector2d, endTangent: Double)
```



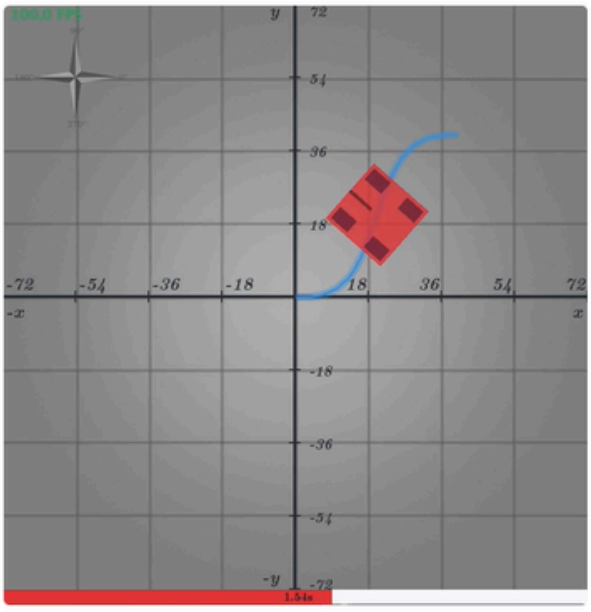
SPLINE A RUMBO LINEAL

```
.splineToLinearHeading(endPose: Pose2d, endTangent: Double)
```



ENCABEZADO SPLINE A SPLINE

```
.splineToSplineHeading(endPose: Pose2d, endTangent: Double)
```



3.4 MEEP MEEP

//¿QUÉ ES UN MEEP MEEP?

Meep Meep es una herramienta de código abierto con todas las funciones para Road Runner que ayuda a los equipos a visualizar y depurar las secuencias de trayectoria que tomarán los robots durante el período autónomo del partido. En el contexto del automóvil, es muy importante establecer con precisión las rutas que seguirá su robot y esta herramienta es muy fácil de usar junto con Road Runner, por eso es popular entre los equipos de la FTC.

Meep Meep proporciona principalmente métodos para seguir rutas y visualizarlas. Esta herramienta tiene el campo del juego que contiene las coordenadas xey que puedes usar como parámetros para los métodos ya creados que te ayudan a generar las rutas deseadas. Es realmente útil para probar una trayectoria, porque inmediatamente después de modificar una coordenada puedes ver los cambios en la pantalla.

Lo guiaré paso a paso a través del proceso de instalación de esta poderosa herramienta.

PROCESO DE INSTALACIÓN

implementó previamente a la izquierda (en la vista del proyecto). Haga clic en nuevo, luego en Módulo.

Cuando aparezca la siguiente ventana, seleccione el tipo de módulo como biblioteca Java y Kotlin, luego nombre la biblioteca exactamente "MeepMeepTesting" y nombre la clase del mismo modo, luego haga clic en finalizar.

Verás que tienes en la ventana izquierda, dentro del proyecto, el módulo MeepMeepTesting con la clase MeepMeepTesting si has hecho todo correctamente.

Lo siguiente que necesita es abrir el script Gradle del módulo, cambiar la versión de Java y copiar un fragmento de GitHub, paso 6 de este enlace (MeepMeep/INSTALL.md en master · rh-robotics/MeepMeep) y pégalo allí. Luego, cuando aparezca "Sincronizar ahora", haga clic en él.

Ahora que tienes el gradle y el módulo, puedes pegar desde el paso 8 desde el mismo enlace ese código en la clase del módulo y para ejecutarlo tendrás que crear una configuración de ejecución. Busque en la barra superior y haga clic en "TeamCode" con el logotipo de Android al lado y luego en Editar configuración. En la esquina superior izquierda, haga clic en + y luego en Aplicación.

Después de hacer clic en Aplicar y Aceptar, ¡listo!