

Programming TeleOp

Getting Started!

This year's Starter Bot program aims to offer flexibility to teams in terms of complexity. This walkthrough will begin with the basic version for Blocks. A version with autonomous built in is also available!

If your team is new to FTC, or in need of a review, we strongly recommend checking out our updated [Hello Robot Programming Guide ↗](#) before diving in!

To begin, let's take a look at the configuration and actuators used in this year's design.

Configuration

Before getting started with programming we needed to create a [configuration file ↗](#). Below is an overview of how the robot is configured for the TeleOp code to function as expected:

Port Type	Port Number	Device Type	Name
Motor	0	REV Robotics Ultraplanetary HD Hex Motor	rightDrive
Motor	1	REV Robotics Ultraplanetary HD Hex Motor	leftDrive
Motor	2	REV Robotics Core Hex Motorr	coreHex
Motor	3	REV Robotics Ultraplanetary HD Hex Motor	flywheel
Servo	0	Continuous Servo	servo

Configuration File Download

Right click the file below and select "save link as". To put this configuration file on your robot, drag this file into the "FIRST" folder of your Control Hub's file system.

 651B

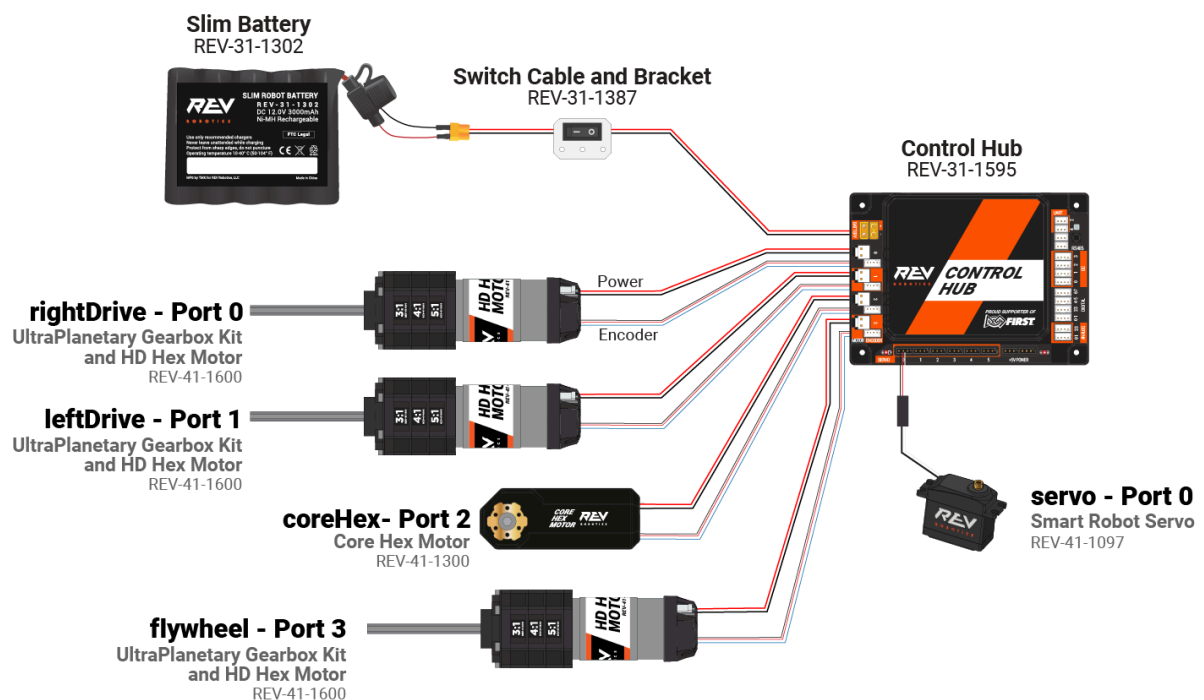
REV Starterbot.xml

 Download


 Open

You will need to activate the configuration using the Driver Hub before proceeding. [Click here to learn more about the configuration process.](#) ↗

Wiring Diagram




Device Name	Device Type	Port
rightDrive	UltraPlanetary Gearbox Kit and HD Hex Motor	Motor/Encoder Port 0
leftDrive	UltraPlanetary Gearbox Kit and HD Hex Motor	Motor/Encoder Port 1
coreHex	Core Hex Motor	Motor/Encoder Port 2
flywheel	UltraPlanetary Gearbox Kit and HD Hex Motor	Motor/Encoder Port 3
servo	Smart Robot Servo - set to CR mode	Servo Port 0

 The Smart Robot Servo can be changed to [continuous mode using the SRS Programmer!](#) ↗

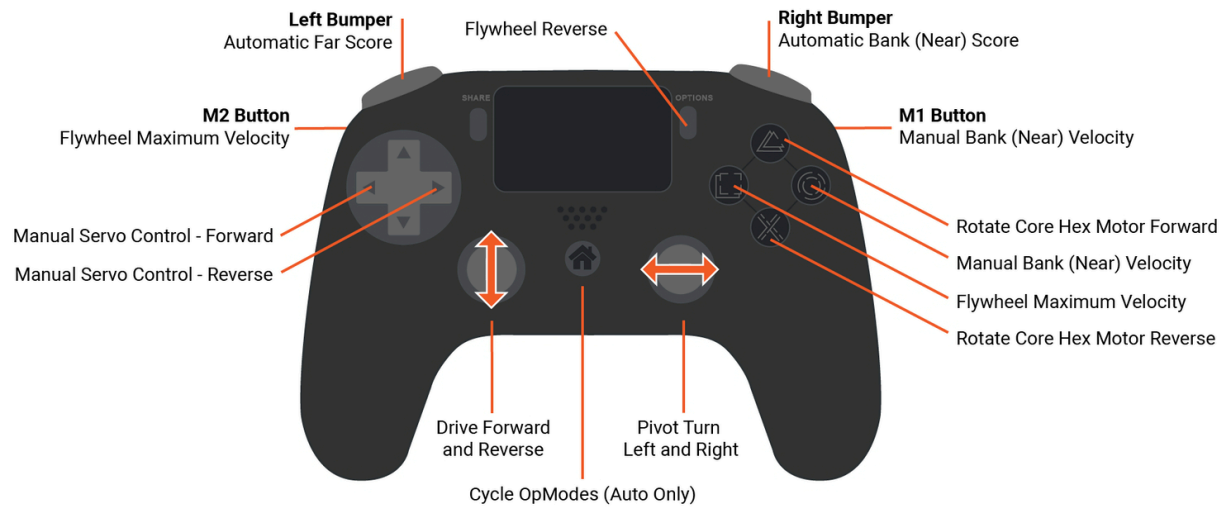
Gamepad Setup

The 2025-26 REV DUO FTC Starter Bot is designed to use a split arcade drive for the default. Check out our Upgrades page for the arcade drive version of the program.

In split arcade drive, the left joystick will control forward and reverse motion of the robot, and the right joystick will control turning. This is similar to how some RC cars are driven and video games are played.

 Not all gamepads have buttons labeled the same way. Check the manufacturer's documentation for accurate button mapping.

2025-26 REV DUO FTC Starter Bot Gamepad Layout:



Gamepad Input	Function
Right Joystick	Turn left and right
Left Joystick	Drive forward and reverse
Home/PS Button	Cycle OpModes during Initialization (Auto code only)
Right Bumper	Activates flywheel, feeder Core Hex, and agitator servo to automatically fire at "bank shot" (near) range
Left Bumper	Activates flywheel, feeder Core Hex, and agitator servo to automatically fire at "far shot" range
Left Dpad/Right Dpad	Manually rotate agitator servo
Y/Triangle	Manually spins the Core Hex at half speed
A/Cross	Manually spins the Core Hex at half speed
B/Circle	Manually spins flywheel to "bank shot" (near) velocity
X/Square	Manually spins flywheel to "max" velocity
M1 (back button)	Manually spins flywheel to "bank shot" (near) velocity
M2 (back button)	Manually spins flywheel to "max" velocity
Options	Runs the flywheel motor in reverse

More information on programming gamepads for use with your robot can be found at [Hello Robot - Using Gamepads ↗](#).


⚠ The REV USB PS4 Compatible Gamepad (REV-31-2983) has two remappable M-buttons on the back side of the controller. For this program, we advise reprogramming them to be the Circle and Square buttons!

[Please see the guide here for how to remap ↗](#).


Basic TeleOp Program Downloads


Blocks:

[Check out *FIRST's* guide on uploading a Blocks program! ↗](#)


 22KB

REVStarterBotTeleOp2025.blk


 Download


 Open

OnBot Java:

 5KB

REVStarterBotTeleOpJava.java

 Download


 Open

TeleOp with Autonomous Program Downloads


Blocks Auto:


- ✓ Please see the [Autonomous Program section of this guide](#) to learn how to switch between the program that will run!

[Check out *FIRST's* guide on uploading a Blocks program! ↗](#)

 55KB

REVStarterBotTeleOpAuto.blk

 Download

 Open

OnBot Java Auto:



10KB

REVStarterBotTeleOpAutoJava.java



Download



Open

Any updates to our example programs are documented in our changelog.

The following pages will dig deeper into unraveling the secrets behind this code. Grab your pickaxe and let's get started!

Quick Links

[Initialization and Variables](#)

[Main Loop](#)

[Flywheel Control](#)

[Manual Control and Drive](#)

[Autonomous Initialization](#)

[Autonomous Code](#)

[OnBot Java Overview](#)

Programming Walkthrough Video

Programming the 2025-2026 REV DUO FTC Starter Bot




Programming Autonomous Video Walkthrough

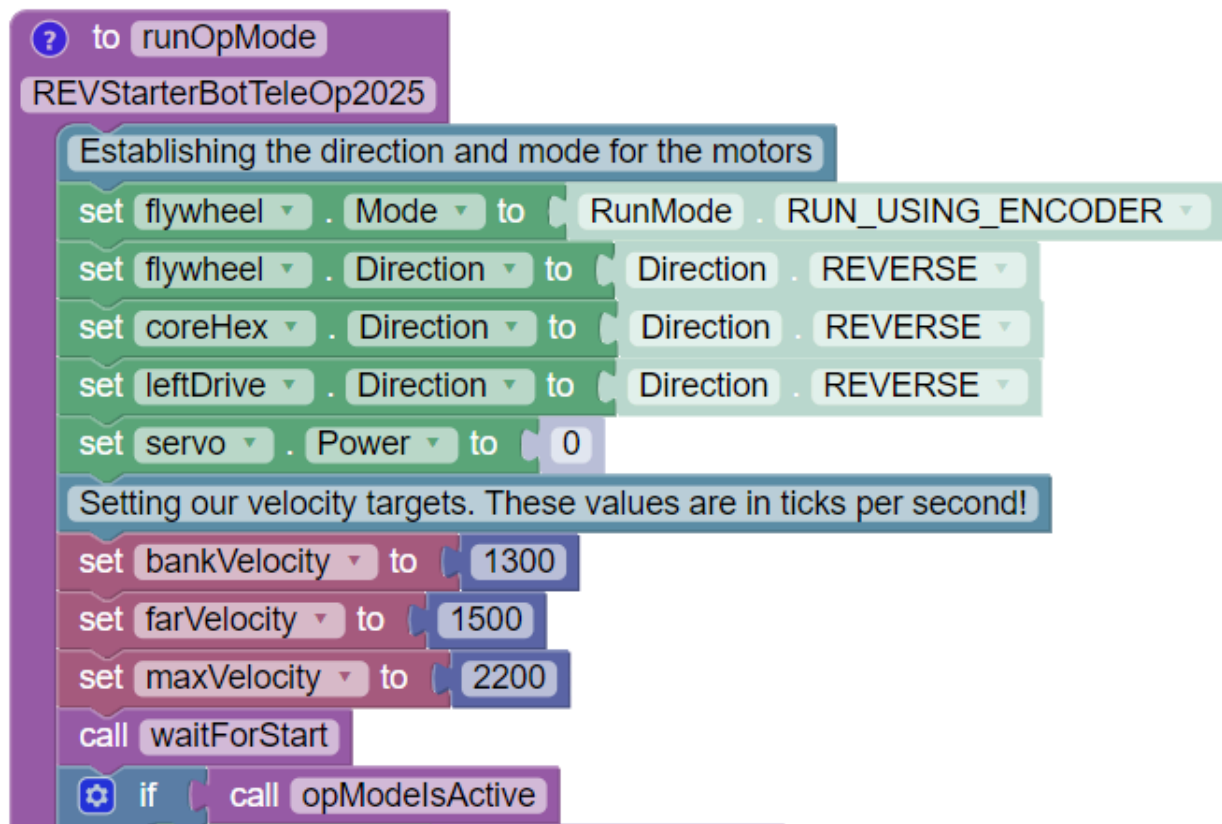
Autonomous Programming: 2025-2026 REV DUO FTC Starter Bot



Programming - Initialization

 This walkthrough is for the example program that DOES NOT include autonomous. For the [autonomous program click here!](#)

Before the bulk of our program begins, we need to first establish our variables and tell our motors how we want them to run. This section of code will run once when the program is activated and before we hit "Play" on the Driver Hub.



Initialization Code

Motor and Servo Settings

Since we are going to be using velocity control for our flywheel, we need to set it up to run with encoders enabled. It has also been set to be reversed due to the direction it rests on our robot. Our Core Hex being used for the feeder is reversed for a similar reason.

Meanwhile the motors on our drivetrain are a mirror of each other, therefore one needs to be set to run in reverse. In this case, we have the leftDrive motor set to run in reverse.

Lastly, our servo is set to no power. Though it's not being used quite yet this helps to make sure it's properly enabled for when it's later called. You may hear it twitch slightly when initializing the program!

Initializing Variables

 [To learn more about "What is a Variable?" check out our Hello Robot tutorial! ↗](#)

The three variables used in our program this year are used to set targets for what velocity our flywheel needs to reach. If at any time the target velocity needs to be adjusted it only has to be updated here to reflect through the whole code!

Variable	Purpose
bankVelocity	The flywheel velocity target for launching balls while against, or close to the goal
farVelocity	The flywheel velocity target for launching balls from a few feet back from the goal
maxVelocity	The "max" velocity for the flywheel. This can be increased, but is not recommended

Variations in your robot build may require these values to be changed. For example, adjusting the deflector may allow the robot to do far shots from further away.

We encourage experimenting with different velocities to determine what works best for your team and robot!

Programming - Main Loop

Main loop for the TeleOp code

The main "whileLoop" of our program this year is fairly short containing our functions and telemetry for the flywheel.

What is a Function?

You can think of functions (also known as methods) as a set of pre-written instructions represented by one line of code. When the function is called, the robot knows to run that set of instructions.

As an example, in our program we've created a function called `splitStickArcadeDrive`. Any time the robot receives an input for that function, in this case from the joysticks, it knows to go through the process of running the drivetrain despite those code lines not being listed individually in the `mainLoop`.

`splitStickArcadeDrive` function being called in the `mainLoop`

Our full `splitStickArcadeDrive` function, and those code steps, can be found elsewhere within our program organized and self-contained with all the relevant pieces together. This code is what's run when the function is called.


`splitStickArcadeDrive` function

In short:

Functions take the place of several lines of code and appear as a single line when called. This can be incredibly useful if there is a section of code we know will be repeated or to break apart our code into chunks for easy editing and viewing.

Below is a breakdown of our functions

Function	Purpose
splitStickArcadeDrive	Contains the code for driving the robot
setFlywheelVelocity	Contains both the auto and manual control for the flywheel
manualCoreHexAndServoControl	Contains manual control for the feeder Core Hex and servo

 Interested in learning more about functions? [Check our Hello Robot!](#) ↗

Telemetry

The telemetry of the main loop reports the flywheel's velocity and how that equals to power. This can be used as a reference when troubleshooting the flywheel or determining if a new velocity target should be set.

Telemetry for the flywheel's velocity and power


Programming - Manual Control and Drive

Split Stick Arcade Drive

Split Stick Arcade Drive

This year's Starter Bot is designed for split arcade drive. This means the left joystick controls the forward and back motion while the right joystick allows for rotation.

The approach to Split Stick Arcade Drive used in this year's Starter Bot is intended to be similar to those available as examples in the SDK from *FIRST* and our tutorial for standard [Arcade Drive](#) ↗!

 To learn more about the variables used and the equation for deciding motor power, check out [Hello Robot's walkthrough](#) ↗!

Manual Feeder and Servo Control

Manual Core Hex and Servo Controls

Manual control is built into the program to allow flexibility in how team's approach launching balls or to aid in the event that a ball becomes stuck.

For visual clarity, the servo and Core Hex controls are separated into separate if/else statements.

Core Hex Feeder

Manual control for the Core Hex feeder

When cross/A is held on the gamepad, the Core Hex feeder will rotate at half power. This would feed balls to the flywheel. While holding triangle/Y, it will rotate at half power in the opposite direction. This would pull balls away from the flywheel back into the hopper.

Be aware both the Core Hex and flywheel motor may need to be reversed to free a ball in the lower area of the launcher.

Servo Agitator

Manual control for the servo agitator

When dpad left or right is held on the gamepad, the agitator servo will continuously spin. This may help with adjusting balls already loaded in the hopper.

Programming - Flywheel Control

All the flywheel controls are contained to the function `setFlywheelVelocity`.

Flywheel if/else statement for auto and manual controls

The flywheel has two main forms of operation:

- **Manual Control**- This will spin up ONLY the flywheel to the target velocity. The driver can then manually spin the Core Hex to feed balls.
- **Auto Control**- This will spin up the flywheel and activate the agitator servo. Once the flywheel is in a specified range of the target velocity, the Core Hex will begin to feed balls automatically. This is intended to be for a fully loaded robot to be able to make multiple shots in quick succession

Let's take a look at our entire if/else statement before exploring our Auto Control options.

Flywheel Control

Breakdown of flywheel buttons

When holding the "**option**" **button** on the gamepad, the flywheel will spin in reverse at half power. This is intended to help with freeing stuck balls if needed.

Reversing the flywheel


The **left bumper** or **right bumper** on the gamepad will request their specified function to run. We'll discuss these more below!

Activating the flywheel launcher's "auto" modes

When pressing the "**circle**" **button**, the flywheel will spin up to the set velocity for a "bank shot". This is the velocity for launching into the goal while against the goal or a couple inches back.

When pressing the "**square**" **button**, the flywheel will spin up to the set "max" velocity. This may be used for launching balls at the goal with adjustments to the deflector, however is intended mostly for teams wanting to explore working with the flywheel and differences in velocity.

Manual control for the flywheel

 Be aware that launching balls at higher velocity can launch them to ceiling height. We do not recommend using this velocity option in rooms with low ceilings or hanging lights.

Turning off Actuators

Setting all actuators to 0 power or velocity

If no button is actively pressed on the gamepad, the flywheel, Core Hex feeder, and servo are told to set power or velocity to 0.

The servo has an additional check to allow manual control on the Dpad to override this function and prevent stuttering in the movements.

Check to prevent the servo from stuttering

Auto Launching with the Flywheel

While a bumper is pressed on the gamepad, the servo, Core Hex feeder, and flywheel will all activate with the intent to launch multiple balls in succession.

Bank "Near" Shot Auto

Code for the bank or "near" shot launching

When **right bumper** is held, the above sequence will run until release. This is intended for launching balls into the goal from against the goal or a couple inches back.

The flywheel is set to spin to the preset "bankVelocity" continuously and the agitator servo will activate.

Lastly, the robot will check first if the velocity of the flywheel is within 50 ticks below the "bankVelocity" before it will allow balls to fire. This value can be adjusted to be a bigger or tighter window if you notice the Core Hex is not feeding as expected.

Check to allow the flywheel to reach velocity between launches

"Far" Power Auto

Far power autonomous launching code

When the left bumper is held, the above sequence will run until release. This is intended for launching balls into the goal from a few feet back from the goal. This may require adjustments to the deflector if teams prefer this approach.

The flywheel is set to spin to the preset "farVelocity" continuously and the agitator servo will activate.

Similar to the bank auto, the robot will run a check to see if the flywheel velocity is currently within a specified window of the target "farVelocity" before it begins feeding. The default range is higher due to the increased velocity, but may be adjusted for further refinement.

- ❗ In the default code example, there is not a check for if the flywheel's velocity is above the "bankVelocity" or "farVelocity" targets. Teams may consider adding this for additional refinement. Keep an eye out on our upgrades page for more information!

Programming - Autonomous Initialization

The autonomous version of the sample program includes the option for both a Blue and Red Alliance Auto, along with TeleOp, all in one code.

Why combine everything into one program?

Combining the Auto codes and TeleOp all together does make the Blocks program appear chunky and intimidating at first glance. However, the goal is for this to prevent issues with updates not being reflected across multiple programs.

For example, say your team decides to make changes to the flywheel that leads to changes in what one of the target velocities should be set to. This change is then updated in TeleOp, perhaps the program the team runs most often, but neither of the auto options. This can lead to frustration and confusion later.

But since everything is combined here, when that one variable is update, it now is automatically reflect in all three options!

Let's take a look first at how we select which OpMode should run when we press "play" on the Driver Hub.

Initialization with Auto

Initialization for the autonomous program

First, you may notice compared to our "default" code that the variables and motor modes are gone. These have been combined into a function, "initRobot", for clarity. We'll look at that function more closely below.

Next, rather than checking if our opModelsActive as we usually do we are instead checking that the program has only been initialized. In this time frame is when we can select which OpMode will run.

Code for switching between OpModes using the gamepad

Our "operationSelected" variable will ultimately set which OpMode will run after receiving input from the function "selectOperation". By default, it is set to run TeleOp if the PS/Home button is never pressed.

The "selectOperation" function will output the end state to change this variable. The PS/Home button on the gamepad will progress the if/else statement used for determining this state. We'll break this down further below!

Function block for inputting operationSelected and gamepad inputs

After "waitForStart" is called, the robot will execute the appropriate function containing the Auto Blue, Auto Red, or TeleOp code based on what was selected.

If/else statement for determining which mode will run

Initializing Variables and Settings

Setting up actuator settings and initializing variables


Let's focus on the new variables added in this version of the code. To learn about the motor settings and existing variables check out [Programming - Initialization](#).

First, we need to set up the variables used while selecting our state to mean something. In this case, rather than the variables being set to a number, we will set them to a name.

Defining the text based variables

Additionally, we need our "operationSelected" to have a default program to run.

Defaulting operationSelected to "TeleOp"

 When making edits to this program, please note that the order the variables are defined MATTERS. If "operationSelected" is defined before the Auto/TeleOp variables are set to a name it will return an error.


For autonomous we'll be using elapsed timers and encoders. Because of this we need to set up some needed information to be referenced later, starting with our timers.

- autoLaunchTimer - controls how long the robot will launch balls during autonomous
- autoDriveTimer - controls how long the robot can drive when backing away from the launch line

Setting up timers for autonomous

For drivetrain encoders, we can go ahead and do our math here needed to convert wheel size into appropriate ticks.

Completing the math needed for using encoders in auto

 Interested in learning more about programming with encoders or elapsed timers? Check out [Hello Robot!](#)

selectOperation Function

Full code for cycling between options using the gamepad

Unlike functions we've used previously, the "selectOperation" function is set up to take in an input then return an output after being completed. In this instance, "state" and "cycleNext" are being inputted then the output will be the variable "state".

Keep in mind the function will repeatedly run while the specified condition is true, in this case so long as the program is set to initialization. Therefore state will be an input and output between each cycle.

Functions such as this are created in Blocks by clicking the gear icon and adding desired inputs.

How to create a function that uses inputs

Cycling OpModes

If/else for cycling between auto blue, auto red, and teleOp OpModes

As mentioned above, our cycleNext is tied to our PS/Home button and if it was pressed (not being held). Each time it is pressed, "state" will progress one step in our if/else statement moving through each option.

In short, our if/else is checking what "state" is currently set to then proceeding to the next option if the button is pressed again. In the event the input is not received properly, a warning will be returned instead as our final "else".

Telemetry Outputs

Telemetry readout to the Driver Hub for what is selected

Based on which "state" or OpMode has been selected, the Driver Hub will showcase different telemetry reads. First is the telemetry to provide the directions for cycling OpModes and the current selection.

Directions for how to cycle options

If an autonomous mode has been selected, then the Driver Hub will read out a prompt to turn on the "Auto timer" built into the Driver Station App for autonomous. This will prevent the robot from continuing to run after the 30 second auto period.

Reminder to turn on the auto timer

Lastly is another instruction for starting the program when ready. You'll see at the bottom of our function the return of "state" to be used.

Directions for starting the program

Programming - Autonomous Code

Once our OpMode is selected, we're ready to actually hit play and let the robot run! For this example, the [TeleOp portion of the code](#) has remained the same so we won't repeat looking at it here.

Instead let's break down our autonomous code options. For both Blue and Red Alliance versions of the code, the robot is intended to start against the goal while touching the launch line. It will automatically fire the pre-loaded balls for 10 seconds before backing up from the goal, turning, and driving straight back off the line.

Auto Blue Alliance

Auto Red Alliance

The two versions of the code are nearly identical with the difference being which direction the robot needs to turn before backing up.

Turn Blue

Turn Red

Let's take a closer look at how our auto code works.

Running Auto

Launching Balls


Code for launching balls for 10 seconds

Our entire auto is contained to an if/else statement checking if `opModelsActive` is true, meaning the "play" button has been pressed on the Driver Hub. Once this occurs, it'll read out that the OpMode is running.

To start launching balls into the target, the associated timer will first be reset. Elapsedtimers begin counting at their creation so it is important to reset them when they're actually going to be used to get a correct time.

With the timer running, the "bankShotAuto", the same used in TeleOp, will begin running for 10 seconds. This time can be adjusted, but was set to 10 seconds in the example as a safe window for teams to begin testing with and observing how their robot acts.

10 second timer for launching in autonomous

 When using a "whileLoop" in Blocks a call of if the "opModelsActive" is required for the loop to properly run.

A telemetry readout to the Driver Hub will show how much time has passed according to the "autoLaunchTimer" to aid with making adjustments.

Telemetry readout for the launcher timer

Once our timer is up, all our actuators will be set back to 0 power or velocity.

Setting actuators back to 0

Driving in Auto

There are two parts to our program for the robot driving in autonomous. In our if/else statement, we have a portion of the "autoDrive" function taking in inputs to calculate the robot's movements. Let's looking at the function itself first.

Full autoDrive code

Here our elapsed timer will reset to be ready for use with the robot's movements. Then the drivetrain motors are given a target position. This target position is calculated using the current position of the motor, the inputted distance in inches from the main function, and our conversion set up during initialization.

Calculating drivetrain movements using the motors' encoders

Next, we're change our motors to "RUN_TO_POSITION" mode for this autonomous. The power to be provided is set by the "speed" given in the main function.

Updating the mode and power for the drivetrain motors

With all that information gathered, our robot will now move in the specified direction and stop at a certain distance when either the motors halt or the timer runs out. Using "call idle" allows our program to progress between the two functions.

Checking that the motors are active and the timer has not run out

Once our movement is complete, our drivetrain motors are set to turn to 0 power and reset back to "RUN_WITHOUT_ENCODER" in preparation for TeleOp.

Updating the mode and power for the drivetrain motors after running

Now let's look at the remainder of our main function of "doAuto".

Movements of the robot in autonomous

For each portion of movement we have inputted:

- **speed**- The set power for the motors
- **leftDistanceInch/rightDistanceInch**- The target distance we want our robot to move to be calculated with our equation. This will be different for each motor when the robot is turning.
- **timeout_ms**- this is the maximum time the elapsetimer can count up to in milliseconds.

Let's take a look at each step of the robot's movement

Back Up

The robot will first back away from the goal, at half power, roughly 12 inches. If something happens where the robot cannot complete this, such as colliding with another robot, this step will timeout after 5 seconds.

Code for the robot backing up

Turn

Next the robot will turn at half power about 6 inches making a 90 degree angle. If something happens where the robot cannot complete this, such as colliding with another robot, this step will timeout after 5 seconds.

Code for the robot turning

Drive Off Line

Finally the robot will back up at full power 50 inches to be off the line. This is much further than needed to be off the line so we encourage teams to adjust this value to get the robot positioned where they prefer.

If something happens where the robot cannot complete this, such as colliding with another robot, this step will timeout after 5 seconds.

Code for the robot driving off the line

Once the robot has finished moving, the code will stop and return to the initialization option.

Programming - OnBot Java Overview

The OnBot Java version of the program is nearly identical to the Blocks version this year. Therefore the walkthrough for the Blocks version is applicable to the OnBot Java option.

Notable differences:

Some methods, also known as functions, have been reorganized for better clarity and readability. For example, the flywheel related methods are organized together.

For autonomous, the initRobot method found in the Blocks version is removed. The related information is called at the beginning when the variables or hardwareMap are first referenced.

Basic OnBot Java Code:


```

package org.firstinspires.ftc.teamcode;

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

@TeleOp
public class REVStarterBotTeleOpJava extends LinearOpMode {

    private DcMotor flywheel;
    private DcMotor coreHex;
    private DcMotor leftDrive;
    private CRServo servo;
    private DcMotor rightDrive;

    // Setting our velocity targets. These values are in ticks per
    second!
    private static final int bankVelocity = 1300;
    private static final int farVelocity = 1900;
    private static final int maxVelocity = 2200;

    @Override
    public void runOpMode() {
        flywheel = hardwareMap.get(DcMotor.class, "flywheel");
        coreHex = hardwareMap.get(DcMotor.class, "coreHex");
        leftDrive = hardwareMap.get(DcMotor.class, "leftDrive");
        servo = hardwareMap.get(CRServo.class, "servo");
        rightDrive = hardwareMap.get(DcMotor.class, "rightDrive");

        // Establishing the direction and mode for the motors
        flywheel.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        flywheel.setDirection(DcMotor.Direction.REVERSE);
        coreHex.setDirection(DcMotor.Direction.REVERSE);
        leftDrive.setDirection(DcMotor.Direction.REVERSE);
        //Ensures the servo is active and ready
        servo.setPower(0);

        waitForStart();
        if (opModeIsActive()) {
            while (opModeIsActive()) {
                // Calling our methods while the OpMode is running
                splitStickArcadeDrive();
                setFlywheelVelocity();
                manualCoreHexAndServoControl();
                telemetry.addData("Flywheel Velocity", ((DcMotorEx)
flywheel).getVelocity());
            }
        }
    }
}

```

```

        telemetry.addData("Flywheel Power", flywheel.getPower());
        telemetry.update();
    }
}

/**
 * Controls for the drivetrain. The robot uses a split stick stlye
arcade drive.
 * Forward and back is on the left stick. Turning is on the right
stick.
 */
private void splitStickArcadeDrive() {
    float x;
    float y;

    x = gamepad1.right_stick_x;
    y = -gamepad1.left_stick_y;
    leftDrive.setPower(y - x);
    rightDrive.setPower(y + x);
}

/**
 * Manual control for the Core Hex powered feeder and the agitator
servo in the hopper
 */
private void manualCoreHexAndServoControl() {
    // Manual control for the Core Hex intake
    if (gamepad1.cross) {
        coreHex.setPower(0.5);
    } else if (gamepad1.triangle) {
        coreHex.setPower(-0.5);
    }
    // Manual control for the hopper's servo
    if (gamepad1.dpad_left) {
        servo.setPower(1);
    } else if (gamepad1.dpad_right) {
        servo.setPower(-1);
    }
}

//Flywheel Code
/**
 * This if/else statement contains the controls for the flywheel,
both manual and auto.
 * Circle and Square will spin up ONLY the flywheel to the target
velocity set.
 * The bumpers will activate the flywheel, Core Hex feeder, and servo

```

to cycle a series of balls.

```
*/  
private void setFlywheelVelocity() {  
    if (gamepad1.options) {  
        flywheel.setPower(-0.5);  
    } else if (gamepad1.left_bumper) {  
        farPowerAuto();  
    } else if (gamepad1.right_bumper) {  
        bankShotAuto();  
    } else if (gamepad1.circle) {  
        ((DcMotorEx) flywheel).setVelocity(bankVelocity);  
    } else if (gamepad1.square) {
```

OnBot Java Code with Auto:


```

package org.firstinspires.ftc.teamcode;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.CRServo;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.util.ElapsedTime;

@TeleOp
public class REVStarterBotTeleOpAutoJava extends LinearOpMode {

    private DcMotor flywheel;
    private DcMotor coreHex;
    private DcMotor leftDrive;
    private CRServo servo;
    private DcMotor rightDrive;

    private static final int bankVelocity = 1300;
    private static final int farVelocity = 1900;
    private static final int maxVelocity = 2200;
    private static final String TELEOP = "TELEOP";
    private static final String AUTO_BLUE = "AUTO BLUE";
    private static final String AUTO_RED = " AUTO RED";
    private String operationSelected = TELEOP;
    private double WHEELS_INCHES_TO_TICKS = (28 * 5 * 3) / (3 * Math.PI);
    private ElapsedTime autoLaunchTimer = new ElapsedTime();
    private ElapsedTime autoDriveTimer = new ElapsedTime();

    @Override
    public void runOpMode() {
        flywheel = hardwareMap.get(DcMotor.class, "flywheel");
        coreHex = hardwareMap.get(DcMotor.class, "coreHex");
        leftDrive = hardwareMap.get(DcMotor.class, "leftDrive");
        servo = hardwareMap.get(CRServo.class, "servo");
        rightDrive = hardwareMap.get(DcMotor.class, "rightDrive");

        // Establishing the direction and mode for the motors
        flywheel.setMode(DcMotor.RunMode.RUN_USING_ENCODER);
        flywheel.setDirection(DcMotor.Direction.REVERSE);
        coreHex.setDirection(DcMotor.Direction.REVERSE);
        leftDrive.setDirection(DcMotor.Direction.REVERSE);
        //Ensures the servo is active and ready
        servo.setPower(0);

        //On initialization the Driver Station will prompt for which OpMode
        should be run - Auto Blue, Auto Red, or TeleOp
        while (opModeInInit()) {

```



```

        operationSelected = selectOperation(operationSelected,
gamepad1.psWasPressed());
        telemetry.update();
    }
    waitForStart();
    if (operationSelected.equals(AUTO_BLUE)) {
        doAutoBlue();
    } else if (operationSelected.equals(AUTO_RED)) {
        doAutoRed();
    } else {
        doTeleOp();
    }
}

/**
 * If the PS/Home button is pressed, the robot will cycle through the
OpMode options following the if/else statement here.
 * The telemetry readout to the Driver Station App will update to
reflect which is currently selected for when "play" is pressed.
 */
private String selectOperation(String state, boolean cycleNext) {
    if (cycleNext) {
        if (state.equals(TELEOP)) {
            state = AUTO_BLUE;
        } else if (state.equals(AUTO_BLUE)) {
            state = AUTO_RED;
        } else if (state.equals(AUTO_RED)) {
            state = TELEOP;
        } else {
            telemetry.addData("WARNING", "Unknown Operation State Reached -
Restart Program");
        }
    }
    telemetry.addLine("Press Home Button to cycle options");
    telemetry.addData("CURRENT SELECTION", state);
    if (state.equals(AUTO_BLUE) || state.equals(AUTO_RED)) {
        telemetry.addLine("Please remember to enable the AUTO timer!");
    }
    telemetry.addLine("Press START to start your program");
    return state;
}

//TeleOp Code

/**
 * If TeleOp was selected or defaulted to, the following will be
active upon pressing "play".
 */

```

```

private void doTeleOp() {
    if (opModeIsActive()) {
        while (opModeIsActive()) {
            // Calling our methods while the OpMode is running
            splitStickArcadeDrive();
            setFlywheelVelocity();
            manualCoreHexAndServoControl();
            telemetry.addData("Flywheel Velocity", ((DcMotorEx)
flywheel).getVelocity());
            telemetry.addData("Flywheel Power", flywheel.getPower());
            telemetry.update();
        }
    }
}

/**
 * Controls for the drivetrain. The robot uses a split stick stlye
arcade drive.
 * Forward and back is on the left stick. Turning is on the right
stick.
 */
private void splitStickArcadeDrive() {
    float X;
    float Y;

    X = gamepad1.right_stick_x;
    Y = -gamepad1.left_stick_y;
    leftDrive.setPower(Y - X);
    rightDrive.setPower(Y + X);
}

/**
 * Manual control for the Core Hex powered feeder and the agitator
servo in the hopper
 */
private void manualCoreHexAndServoControl() {
    // Manual control for the Core Hex intake
    if (gamepad1.cross) {
        coreHex.setPower(0.5);
    } else if (gamepad1.triangle) {
        coreHex.setPower(-0.5);
    }
    // Manual control for the hopper's servo
    if (gamepad1.dpad_left) {
        servo.setPower(1);
    } else if (gamepad1.dpad_right) {
        servo.setPower(-1);
    }
}

```

```

    }

    /**
     * This if/else statement contains the controls for the flywheel,
     both manual and auto.
     * Circle and Square will spin up ONLY the flywheel to the target
     velocity set.
     * The bumpers will activate the flywheel, Core Hex feeder, and servo
     to cycle a series of balls.
     */
    private void setFlywheelVelocity() {
        if (gamepad1.options) {
            flywheel.setPower(-0.5);
        } else if (gamepad1.left_bumper) {
            FAR_POWER_AUTO();
        } else if (gamepad1.right_bumper) {
            BANK_SHOT_AUTO();
        } else if (gamepad1.circle) {
            ((DcMotorEx) flywheel).setVelocity(bankVelocity);
        } else if (gamepad1.square) {
            ((DcMotorEx) flywheel).setVelocity(maxVelocity);
        } else {
            ((DcMotorEx) flywheel).setVelocity(0);
            coreHex.setPower(0);
            // The check below is in place to prevent stuttering with the
            servo. It checks if the servo is under manual control!
            if (!gamepad1.dpad_right && !gamepad1.dpad_left) {
                servo.setPower(0);
            }
        }
    }

    //Automatic Flywheel controls used in Auto and TeleOp

    /**
     * The bank shot or near velocity is intended for launching balls
     touching or a few inches from the goal.
     * When running this function, the flywheel will spin up and the Core
     Hex will wait before balls can be fed.
     * The servo will spin until the bumper is released.
     */
    private void BANK_SHOT_AUTO() {
        ((DcMotorEx) flywheel).setVelocity(bankVelocity);
        servo.setPower(-1);
        if (((DcMotorEx) flywheel).getVelocity() >= bankVelocity - 100) {
            coreHex.setPower(1);
        } else {
            coreHex.setPower(0);
        }
    }

```

```

    }
}

/**
 * The far power velocity is intended for launching balls a few feet
 * from the goal. It may require adjusting the deflector.
 * When running this function, the flywheel will spin up and the Core
 * Hex will wait before balls can be fed.
 * The servo will spin until the bumper is released.
 */
private void FAR_POWER_AUTO() {
    ((DcMotorEx) flywheel).setVelocity(farVelocity);
    servo.setPower(-1);
    if (((DcMotorEx) flywheel).getVelocity() >= farVelocity - 100) {
        coreHex.setPower(1);
    } else {
        coreHex.setPower(0);
    }
}

//Autonomous Code
//For autonomous, the robot will launch the pre-loaded 3 balls then
//back away from the goal, turn, and back up off the launch line.

/**
 * For autonomous, the robot is using a timer and encoders on the
 * drivetrain to move away from the target.
 * This method contains the math to be used with the inputted
 * distance for the encoders, resets the elapsed timer, and
 * provides a check for it to run so long as the motors are busy and
 * the timer has not run out.
 */
private void autoDrive(double speed, int leftDistanceInch, int
rightDistanceInch, int timeout_ms) {
    autoDriveTimer.reset();
    leftDrive.setTargetPosition((int) (leftDrive.getCurrentPosition() +
leftDistanceInch * WHEELS_INCHES_TO_TICKS));
    rightDrive.setTargetPosition((int) (rightDrive.getCurrentPosition()
+ rightDistanceInch * WHEELS_INCHES_TO_TICKS));
    leftDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    rightDrive.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    leftDrive.setPower(Math.abs(speed));
    rightDrive.setPower(Math.abs(speed));
    while (opModeIsActive() && (leftDrive.isBusy() ||
rightDrive.isBusy()) && autoDriveTimer.milliseconds() < timeout_ms) {
        idle();
    }
    leftDrive.setPower(0);
    rightDrive.setPower(0);
}

```

```

    rightDrive.setPower(0);
    leftDrive.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
    rightDrive.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
}

/**
 * Blue Alliance Autonomous
 * The robot will fire the pre-loaded balls until the 10 second timer
ends.
 * Then it will back away from the goal and off the launch line.
 */
private void doAutoBlue() {
    if (opModeIsActive()) {
        telemetry.addData("RUNNING OPMODE", operationSelected);
        telemetry.update();
        // Fire balls
        autoLaunchTimer.reset();
        while (opModeIsActive() && autoLaunchTimer.milliseconds() <
10000) {
            BANK_SHOT_AUTO();

```