

# Composite controllers - RobotDrive

The **RobotDrive** class is designed to simplify the operation of the drive motors based on a model of the drive train configuration. The program describes the layout of the motors. Then the class can generate all the speed values to operate the motors for different configurations. For cases that fit the model, it provides a significant simplification to standard driving code. For more complex cases that aren't directly supported by the **RobotDrive** class it may be subclassed to add additional features or not used at all.

## Create a RobotDrive object with 2 motors

```
RobotDrive drive(1, 2);    // left, right motors on ports 1,2
```

First, create a **RobotDrive** object specifying the left and right Jaguar motor controllers on the robot, as shown.

## Creating a RobotDrive object with 4 motors

```
RobotDrive drive(1, 2, 3, 4); // four motor drive configuration
```

In this case, for a four motor drive all the motors are specified in the constructor.

## Creating a RobotDrive object using speed controllers that are already created

By default, the **RobotDrive** object created with port numbers as shown in the previous two examples will allocate Jaguar speed controller objects for each of the motors. If the **RobotDrive** object creates the speed controllers, then it will also be responsible for deleting them when the **RobotDrive** object is deleted.

In some case (as shown here) you might want to be in control of the speed controller objects, for example, at times your program might have a need to operate them independently from the **RobotDrive** object. Another case is if your robot is not using Jaguar speed controllers. In this case, allocate the desired speed controller objects and pass them as parameters to the constructor. Your program will be responsible for deleting the objects when you are done using them.

# Composite controllers - RobotDrive

## Operating the motors of the RobotDrive

Once set up, there are methods that can help with driving the robot either from the Driver Station controls or through programmed operations. These methods are described in the table below.

**Drive(speed, turn)** - Designed to take speed and turn values ranging from - 1.0 to 1.0. The speed values set the robot overall drive speed; with positive values representing forward and negative values representing backwards. The turn value tries to specify constant radius turns for any drive speed. Negative values represent left turns and the positive values represent right turns.

**TankDrive(leftStick, rightStick)** - Takes two joysticks and controls the robot with tank steering using the y-axis of each joystick. There are also methods that allow you to specify which axis is used from each stick.

**ArcadeDrive(stick)** - Takes a joystick and controls the robot with arcade (single stick) steering using the y-axis of the joystick for forward/backward speed and the x-axis of the joystick for turns. There are also other methods that allow you to specify different joystick axes.

**HolonomicDrive(magnitude, direction, rotation)** - Takes floating point values, the first two are a direction vector the robot should drive in. The third parameter, rotation, is the independent rate of rotation while the robot is driving. This is intended for robots with 4 Mecanum wheels independently controlled.

**SetLeftRightMotorSpeeds (leftSpeed, rightSpeed)** - Takes two values for the left and right motor speeds. As with all the other methods, this will control the motors as defined by the constructor.

## Inverting the sense of some of the motors

```
SetInvertedMotor(kFrontLeftMotor, true);
```

It might turn out that some of the motors used in your RobotDrive object turn in the opposite direction. This often happens depending on the gearing of the motor and the rest of the drive train. If this happens, you can use the SetInvertedMotor() method, as shown, to reverse a particular motor.