

Tank Drive: Principles and Programming

Please complete the Control System Overview and Intro to FTC Programming tutorials first.

There are several different types of Tank-drives. This tutorial will cover the basic different types of tank drivetrains and go over some examples and the pros/cons of why you might use one over the other. The big difference in tank drivetrains, compared to holonomic drivetrains (like mecanum wheels) is that tank drives cannot strafe (move side to side laterally). Tank drives are slightly simpler to program and often less expensive. They sacrifice maneuverability, but they can provide greater traction and stability and can be excellent for rough terrain.

<https://gm0.org/en/latest/docs/common-mechanisms/drivetrains/index.html>

This tutorial will cover basic TeleOp programming for tank drives. We will explore 2 different ways to configure your controller.

This tutorial shows only a few options of how to program your robot. There are many different ways to do it. The methods shown here are ideal for new programmers just starting out. But, they will work well!

4 Wheel/2 Motor Configuration:

The most basic type of tank drive is a 2 motor/4 wheel configuration. This is usually only used in very beginner robots. Its only advantages are that it uses fewer motors and is cheaper to build. An example of this is the Rev EDU Kit V2.

https://www.revrobotics.com/rev-45-2041/?searchid=4537596&search_query=kit+bot
<https://docs.revrobotics.com/duo-control/hello-robot-java/part-2/programming-drivetrain-motors>



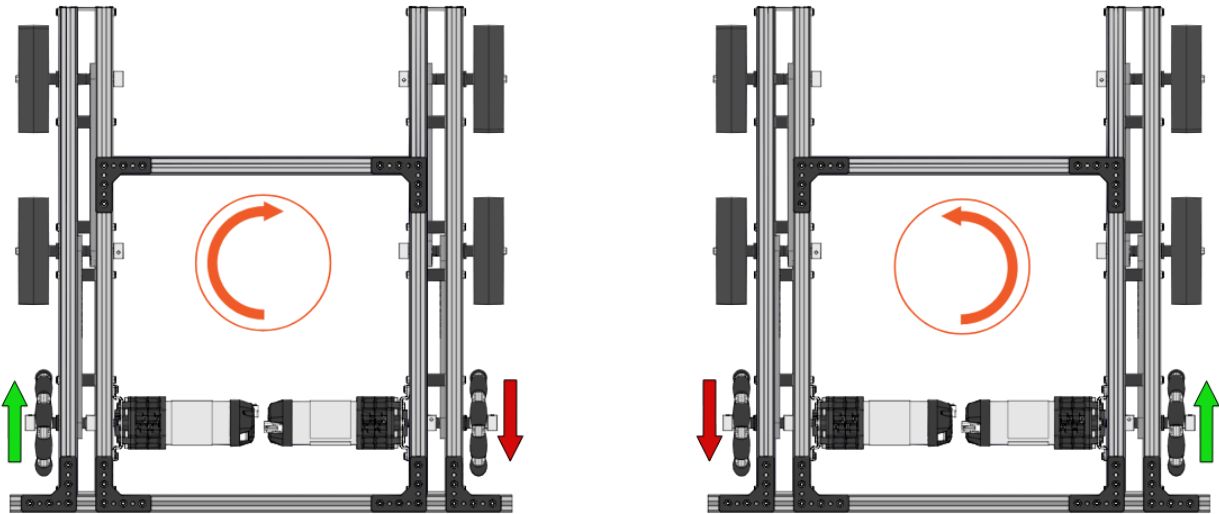
*This configuration has 2 wheels that are connected to motors and 2 wheels that are not. The 2 wheels connected to the motors are generally some kind of standard traction wheel. The two wheels that are NOT connected to the motors are generally omni wheels that can move side to side on their rollers, as well as back and forth like standard traction wheels. This allows the non-motorized wheels to move freely in whichever direction the other wheels dictate, without dragging.

How movement is controlled:

Whether you have just one motor per side, or several motors per side, all tank drives operate basically the same way. If there is more than 1 motor on a given side, the motors work together to power all the wheels in the same direction at the same time on each side.

If you want to go forward, you have your left and right motors/wheels go forward at the same time. If you want to go backward, you have your left and right motors/wheels go backward at the same time.

If the motors/wheels on the left go forward at the same time that the motors on the right go backwards, the robot will turn clockwise (to the right) along a central pivot point. If the motors/wheels on the right go forward at the same time as the motors on the left go backwards, the robot will turn to the left along a central pivot point.



There are 2 common ways to program a basic tank-drive robot: Skid Steer and Arcade. Either way is valid- this is a matter of preference. It might be a good idea to try them both out and see which one your drivers prefer.

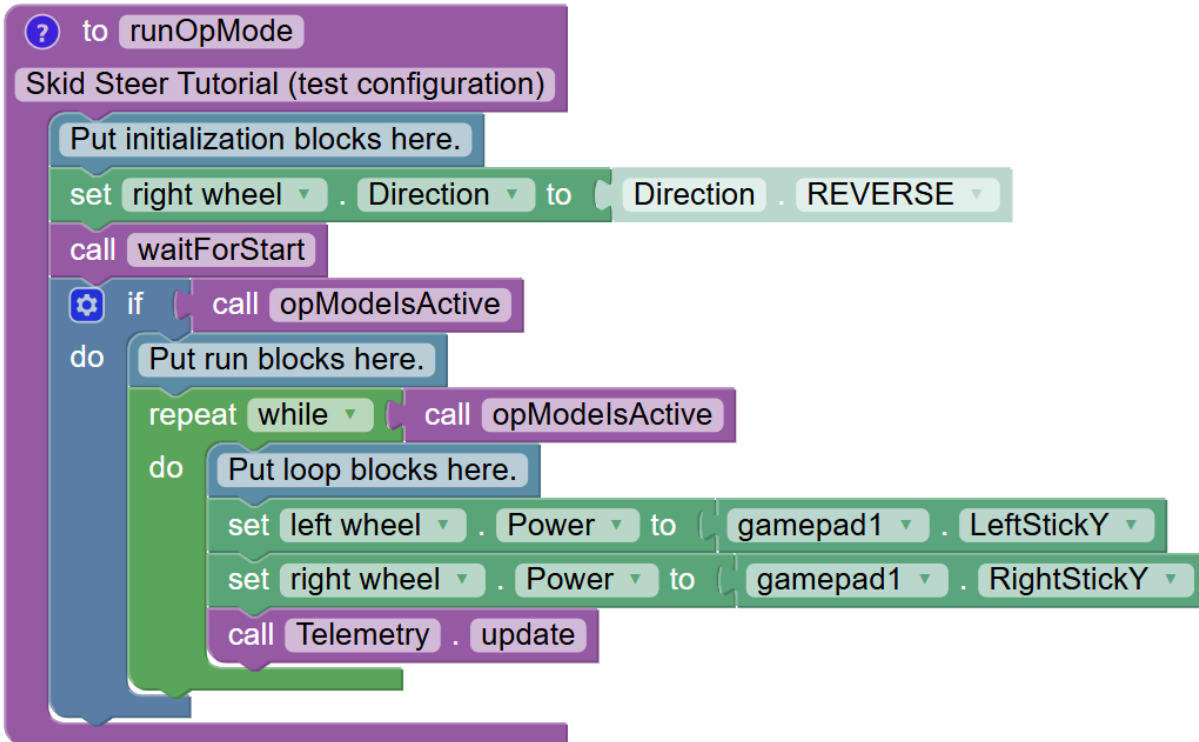
Skid Steer:

In a Skid Steer setup, the left joystick Y axis controls the left side of the robot and the right joystick Y axis controls the right side of the robot. To go forward, you push up/forward on the left and right joysticks at the same time. You push down on both joysticks to go backward.

To turn the robot to the left, you push up/forward on the right joystick and down/backward on the left joystick at the same time. To turn the robot to the right, you push up/forward on the left joystick and down/backward on the right joystick at the same time.

*If you engage the motors/wheels on only one side, it will also turn, but the pivot point will be around whichever wheel on the non-moving side has the most traction with the ground. This kind of movement is only possible with skid steer programming (not arcade). Getting used to driving a robot with skid steer is a little difficult at first for most people, but it's easy to get the hang of it with practice.

Skid Steer TeleOp:



The first step is to name your motors in your configuration. Make sure to name them something that you will be able to easily recognize. They have been named “left wheel” and “right wheel” in this example.

-Create a new OpMode and name it. This OpMode is named “Skid Steer Tutorial (test configuration)”. You may store several different configurations on your robot and activate different ones at different times. The parentheses “(test configuration)” has been included to indicate which robot configuration should be used for this OpMode.

-Reverse one of the wheel directions in the initialization section. If the motors are in a mirror image configuration to each other (which they normally are), then the motor on one side will need to be reversed from the motor on the opposite side. Figuring out which motor needs to be reversed may take some trial and error. The block for **“set.right wheel.direction to. Reverse”** can be found under the **“DcMotor”** tab in the **“Actuators”** drop-down. You can set all the motors to either “forward” or “reverse” in the initialization section to help you keep track. But, if you don’t designate a direction, it will default to “forward.” The above example does not set the direction for the “left wheel” motor, so it is set to “forward” as a default.

-In the loop blocks section, set the left wheel motor power to the left stick Y axis. Set the right motor wheel power to the right stick Y axis. (See example above). The “**set.left wheel.power to**” block can be found in the “**Actuators**” menu as well. The “**Gamepad1. LeftstickY**” block is found in the “**Gamepad**” menu. The blocks can be copied and the drop-down menus can be used to change “right wheel” to “left wheel” and to change “LeftStickY” to “RightStickY”

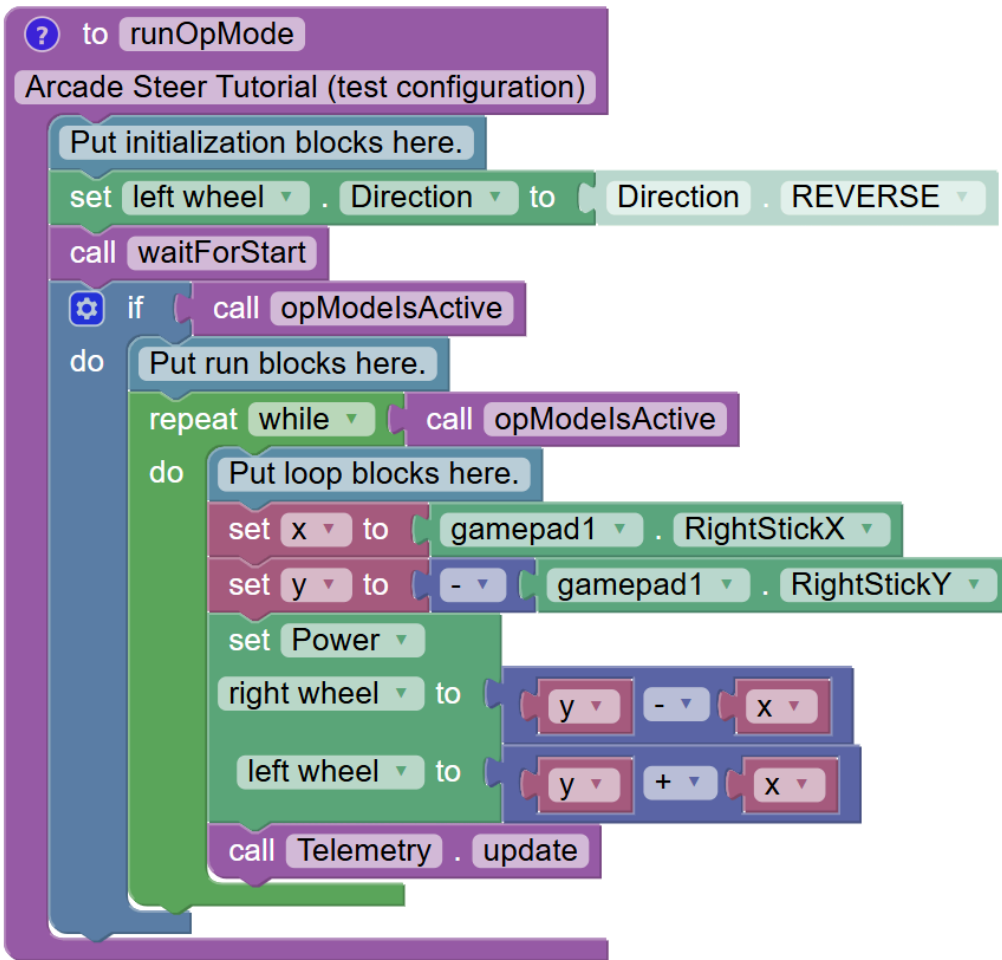
That’s it- don’t forget to save!

Arcade TeleOp:

In Arcade mode, all of the motion is controlled with a single joystick. If you push the right joystick up/forward, the left AND right wheel will both move forward together. If you push the right joystick down/backward, the left AND right wheel will both move backward together.

To turn the robot to the left (counterclockwise), you push the right joystick to the left. To turn the robot to the right (clockwise), you push the right joystick to the right.

This frees up the left joystick for other functions on the robot. It may also be more comfortable for drivers, as it is a more intuitive feeling configuration for many people.



We did not have to confront this in the skid steer example, but it is important to know that a standard video game controller joystick's Y axis is reversed from what we usually think it should be intuitively. Pushing the joysticks up/forward corresponds to negative numbers. So, we often have to reverse the direction of the Y axis in our instructions in order to make our controller logic work correctly with more complicated driving programs. We will see how this fact changes our code as we go through this example.

-First, we reverse one of our motor's direction in the initialization section, just like we did with the skid steer OpMode. In this example, we are reversing the "left wheel" motor instead of the "right wheel" motor as we did in the skid steer OpMode.

-Next, we will create some variables. Creating variables is a great way to keep ourselves organized. We could have written this code without the variables, but this is the recommended way to do it and it is a good habit to start building code this way. A variable is just a placeholder. You can set it to mean whatever you want it to mean. We

will create 2 variables for this code. In this example, we have named them “x” and “y.” You can name them whatever you want, but make sure it is descriptive of what it is representing. In this case, they are representing the x-axis on the right joystick and the y-axis on the right joystick. To create the variables, open the “**Variables**” menu and click “**create variable.**” Once you create a new variable, new blocks will appear for you to work with.

-Set your variables in the loop blocks (green “repeat while” loop). Set your “x” variable to the right joystick x axis. The “**set.x.to.gamepad1.RightStickX**” blocks can be found in the “**Variables**” and “**Gamepad**” menus. Next, set “y” to the right joystick y axis. However, you have to add a negative modifier before the “**gamepad1.RightStickY**” block. This is because, as we mentioned above, the Y-axis on a standard gamepad is reversed from what we would like it to be for the purposes of programming our robot. You can find the “-” (negative) block in the “**Math**” menu.

**Now, we can see why we had to change which wheel we set to reverse in arcade OpMode compared to skid steer OpMode. Basically, in the skid steer example, we set the “wrong” wheel to reverse. In the skid steer OpMode program, we could have set the left Y axis and right Y axis to negative and switched which wheel was reversed to match what we are doing here in the Arcade OpMode. But, we didn’t do that... because it’s best to simplify things sometimes. If this is a little confusing, don’t worry. Just remember that the Y-axis is reversed and that you may have to do some trial and error to figure out which motor/motors need to be reversed in the initialization section.*

-Now that we have set our variables, we can set up our motor power logic in the loop blocks section (green “repeat while”) loop. You can find the blocks that have 2 different power instructions on the same block in “**Actuators**” in the “**DcMotor**” dropdown, then in the “**Dual**” further subcategory.

We set the power of the left and right motor in this block. The power of the right wheel should be “y minus x.” The power of the left wheel should be “y plus x.” This logic may not be intuitive at first, but you can think through it to confirm how it works.

Example: If you push the right joystick to the left all the way, you are giving the left axis a power of -1 and the y axis a power of 0. Plugging those numbers into our motor power formula, the right wheel will get a power of “zero minus negative one.” This equals 1. So, our right wheel gets a power of 1 and will move forward at full power. Our left wheel will get a power of “zero plus negative one.” This means the left wheel gets a power of -1 and will go backwards at full power. If the left wheels are moving backwards and the right wheels are moving forward, the robot will turn left as we hoped. You can

work through this with other examples if desired, but testing it out and seeing how it works in the robot is always the best way to prove that it works!

You can find the blue box that you plugged the “x” and “y” variables into under the “**Math**” menu.

4 Wheel/4 Motor Configuration:

In this configuration, each of the 4 wheels is powered by its own motor. You can use traction wheels or omni wheels, but it is generally recommended to still do 2 traction wheels and 2 omni wheels for the best control and smoothest motion. All traction makes steering rough. All omni makes it a little slippery. Try to position the wheels close to a square for smoothest handling.

This configuration has more power and more traction than the 4 Wheel/2 Motor configuration. It is great for uneven terrain, especially if the body of the robot is kept high off the ground. This configuration can be a little tippy though, so you want to make sure your weight is well distributed and kept in the center as much as possible.

Programming:

The programming for a 4 Wheel/4 Motor Configuration follows the same logic as the 4 Wheel/2 Motor Configuration, but you have to make sure both left motors are moving together and both right motors are moving together (identical to the examples shown below in the BeeLine 6 Wheel demo).

6 Wheel Configurations:

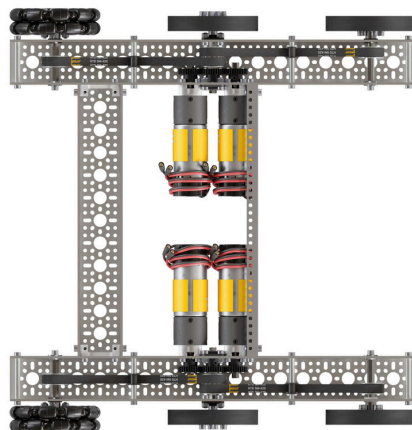
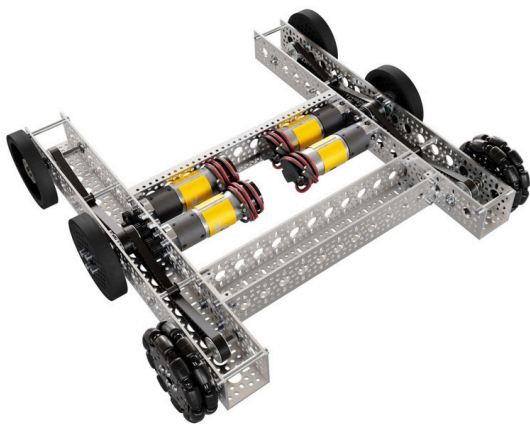
It is possible to build a 6-wheel configuration where only 2 or 4 of the wheels are being powered by motors, but most 6-wheel configurations are set up so that all 6 wheels are getting power from a motor. However, usually you are only using 4 motors to power all 6 wheels. One or two robots can power more than 1 wheel at once using gears and chains or belts. Using 2 motors per side is helpful, because it will give you more power than just one motor per side.

A 6-wheel configuration is great for traction and stability. However, dragging while turning is a huge issue with this configuration. There are 2 ways to solve this issue.

- 1) Drop the center wheels slightly lower than the corner wheels. This will make it so that only 4 of the wheels are touching the ground at any given time (based on the distribution of the robot's weight and the way you are turning). This eliminates the drag of the extra wheels.
- 2) Corner omni wheels: By putting omni wheels on the corners, the wheels can move laterally, and drag is reduced. This configuration can reduce traction somewhat, however.

We will use the **GoBilda BeeLine Chassis Kit** for our example. It is a 4 motor/6 wheel configuration. This kit has paired omni wheels at one of the corner positions and also has a dropped center wheel. 2 motors are used per side. The two motors are linked together with gears, so they essentially operate as one extra powerful motor. The 2 linked motors control all 3 wheels on each side with belts. So, when the motors move, all 3 wheels on either side move forward or backward the same amount.

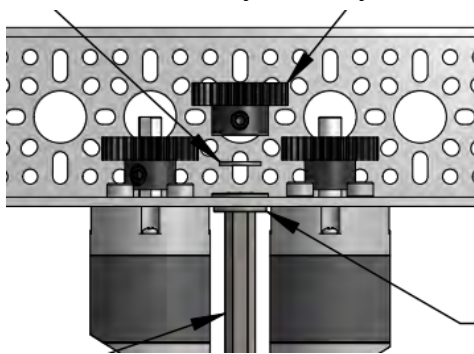
GoBilda BeeLine Chassis Kit V2: <https://www.gobilda.com/beeline-chassis-kit-v2/>



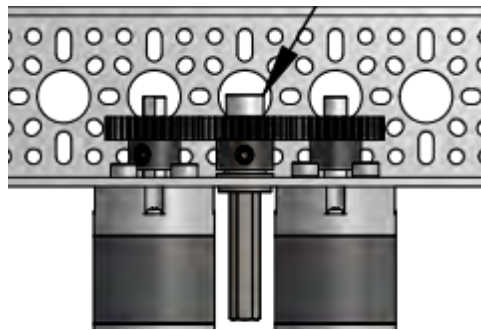
Programming:

The concepts of programming the robot is very similar to the 4 Wheel/2 Motor configuration. But, you have to make sure you have the correct motors set to “forward” and “reverse” in the initialization section.

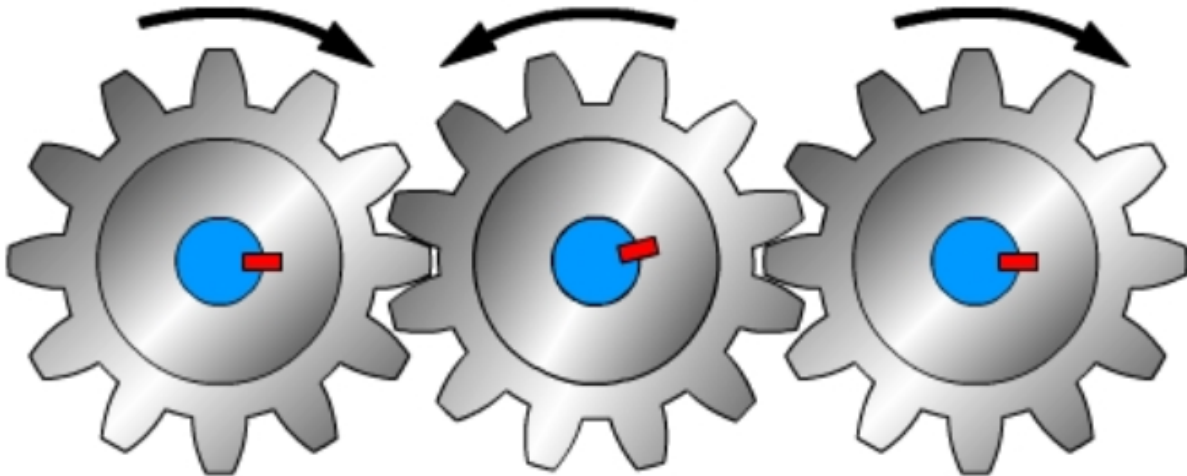
In the GoBilda BeeLine chassis kit, 2 motors are used per side. Each motor turns a gear and both of those gears work together to turn a central gear. This central gear is then responsible for moving all 3 wheels on that side with belts. *There is some variation in this by which year’s model you have, but the same principle applies.



Central gear being installed



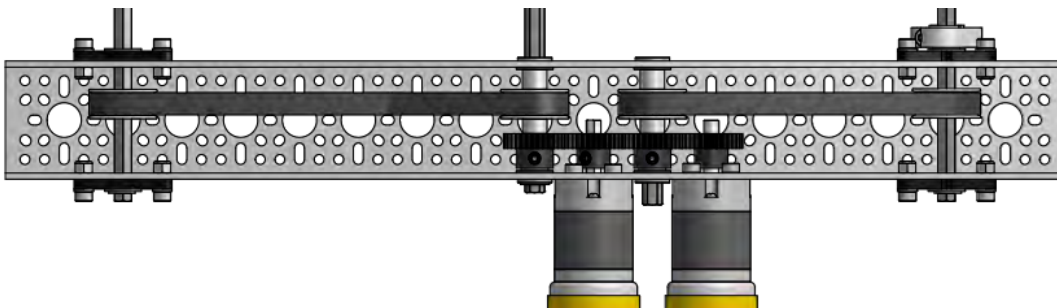
Central gear locked into motor gears



This is a visualization of the gear train that is being used here. The 2 gears on the outside of this gear train are being powered by our 2 motors. The gear in the middle is determining the direction of our belts and wheels. You can see that the 2 gears on the outside are moving in the SAME direction, which is opposite to the direction that the central gear (and therefore our wheels) are moving.



Example 1: *Wheels being powered by belts off central gear*



Example 2: *Wheels being powered by belts off central gear and 4th gear*

Make sure that the 2 motors on the right side are moving in the same direction as each other (either “forward” or “reverse”) and that the 2 left motors are both moving in the same direction as each other. Furthermore, the 2 right motors should be in the opposite direction as the 2 left motors (because they are set up in a mirror image configuration to each other). Once you have finished your base code, you can play around with which side to make “forward” and which side to make “reverse.” This change will flip which side is the “front” of your robot.

*It is possible that a configuration with 2+ motors per side might not all be set to the same direction. It depends on how they are linked to each other and to the belts and gears.

*Notes: The GoBilda motor cords have a connector in the middle. You can reverse the polarity of the motor power by switching how the cord is plugged in. This can be handy if you have installed something incorrectly. But, try to make sure that the red wire is matched to the red wire and the black is matched to the black when you first get started. If any motor's polarity is switched (by switching the wires), then you will have to reverse that motor from “forward” to “reverse” or from “reverse” to “forward” in your programming.



Skid Steer TeleOp:

*GoBilda BeeLine Chassis (2 motors per side, linked together by gears)

```
to runOpMode
  Skid Steer Tutorial BeeLine (test configuration)
  Put initialization blocks here.
  set right front . Direction to Direction . FORWARD
  set right back . Direction to Direction . FORWARD
  set left front . Direction to Direction . REVERSE
  set left back . Direction to Direction . REVERSE
  call waitForStart
  if call opModelsActive
  do
    Put run blocks here.
    repeat while call opModelsActive
    do
      Put loop blocks here.
      set Power
      right front to gamepad1 . RightStickY
      right back to gamepad1 . RightStickY
      set Power
      left front to gamepad1 . LeftStickY
      left back to gamepad1 . LeftStickY
      call Telemetry . update
```

This program is very similar to the 4 Wheel/2 Motor Skid Steer Program. There are 2 changes. 1) We have identified the direction for all 4 motors. All 4 motors have been included here. Even though we could have just left the 2 right motors blank, it is helpful to have them all here to keep track of them, and change them quickly if needed. 2) We have set the power of BOTH the right motors to the right joystick Y axis and BOTH the left motors to the left joystick Y axis.

Arcade TeleOp:

*GoBilda BeeLine Chassis (2 motors per side, linked together by gears)

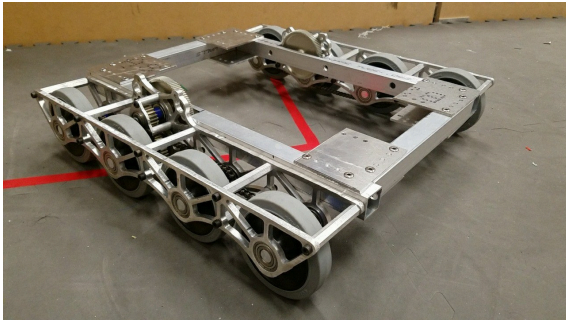
```
to runOpMode
  Arcade Tutorial BeeLine (test configuration)
  Put initialization blocks here.
  set right front . Direction to Direction . REVERSE
  set right back . Direction to Direction . REVERSE
  set left front . Direction to Direction . FORWARD
  set left back . Direction to Direction . FORWARD
  call waitForStart
  if call opModelsActive
  do Put run blocks here.
  repeat while call opModelsActive
  do Put loop blocks here.
  set x to gamepad1 . RightStickX
  set y to - gamepad1 . RightStickY
  set Power
  right front to y - x
  right back to y - x
  set Power
  left front to y + x
  left back to y + x
  call Telemetry . update
```

Once again, the only difference between this code, and the 4 Wheel/2 Motor configuration is the addition of the correct direction for all 4 motors in the initialization

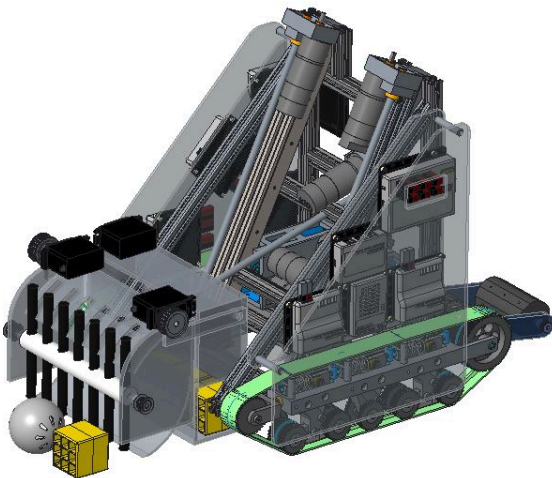
section, and the inclusion of all 4 motors in the loop blocks section. We have applied the same logic for both right motors ($y - x$) and both left motors ($y + x$).

8 Wheel and Tread Configurations:

An 8 Wheel Configuration works in a very similar way to a 6 Wheel Configuration. Generally all 4 center wheels are slightly dropped to reduce turning friction. They have slightly more agility than 6 wheel configurations, but they can be fussy.



Treads can be used on each side instead of wheels. This configuration is rarely used in competition. It is excellent for pushing things and traversing uneven terrain. They are very difficult to get right.



Programming:

The principles for programming are very similar to those for 6 wheel configurations.

Troubleshooting:

Oh no! My code isn't working! This is common. One small error can throw everything off. The error might be in the physical robot, the wiring, the power or the code. Here is a list of items to check when troubleshooting your code.

- 1) Are my wheel motors plugged in completely?
 - Is the encoder cord plugged in completely?
 - Is each motor and encoder plugged into the correct slot on my control hub or control hub extension (according to what is listed on my robot's configuration)?
 - Do I have the correct configuration selected?
 - Some motors can have their direction changed by flipping a switch or changing how their cord is plugged in in the middle (GoBilda). Is this on the correct setting/plugged in correctly?
- 2) Is there any physical object stopping my robot from moving? This could be an internal component, like something stuck in the gear, etc?
- 3) Do I have all of my wheels set to "forward" or "reverse" correctly?
- 4) Did I select all 4 different motors in my "set power to" section?
- 5) Did I double check the + and - symbols on my arcade mode logic?
- 6) Did I remember to multiply the Y-axis variable by "-" when I set my "y" variable in arcade mode? Gamepad Y-axis is reversed from what I want.
- 7) Are all of my instructions inside the green "**repeat while.call. OpModeisActive.do**" box?
- 8) Did I select the correct TeleOp from my drop-down menu on my driver station?
- 9) Is my robot's battery running low?
- 10) My robot is running too fast/really jerky! See "Slow Down the Robot" in the Optional Modifications section below.

Optional Modifications:

Slow Down the Robot:

One common issue is that your robot is running too fast. If it feels jerky and you are struggling to make fine-tuned motions, you may want to slow your robot down. This can be done by multiplying our motor power by an amount that is <1 . This is usually done by adding a variable called "power gain."

https://www.youtube.com/watch?v=rpfhDDX_LOs