

Mecanum Drive: Principles and Programming

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

Mecanum wheels are one of the major types of wheels used in First Tech Challenge Robotics.

-They offer the ability to make precise motions, including strafing (moving laterally)

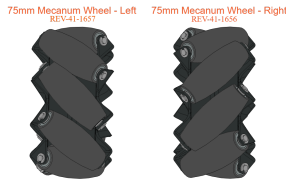


Image of Rev Mecanum Wheels (left and right).

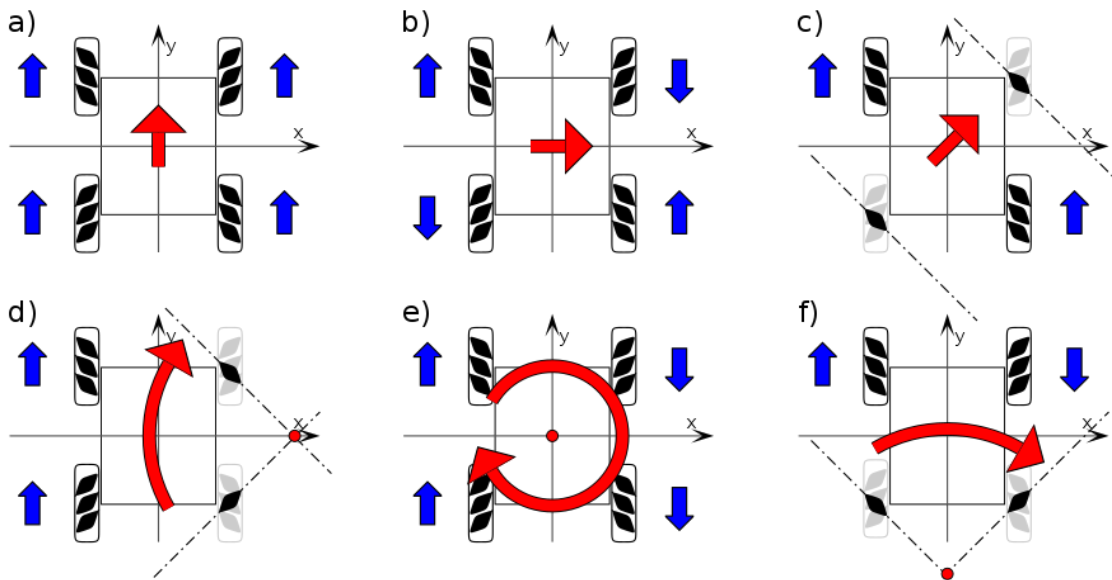


Mecanum wheels on URANUS mobile robot.

**It is important that the mecanum wheels are placed in the correct orientation on the robot!*

This graph summarizes what all 4 wheels will be doing to achieve the desired movement:

-Ex: If you want to strafe right, the front left and back right wheel must move forward while the front right and back left back wheels move backward. Another way to think of this is that the wheels that are “in front” in the direction you want to move must move “inward” toward the robot’s center while the wheels that are “in back” in the direction you want to go must move “outward” from the robot’s center.



Learn More/Sources:

Rev Duo Mecanum Wheels Information:

<https://docs.revrobotics.com/duo-build/mecanum-drivetrain-kit-mecanum-drivetrain/mecanum-wheel-setup-and-behavior>

Example Code in blocks (Rev Duo)

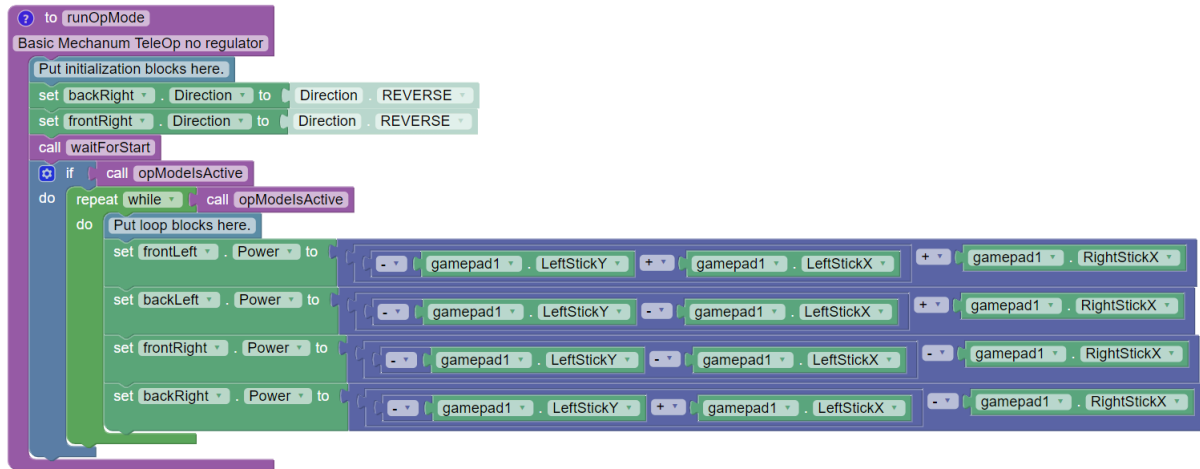
<https://docs.revrobotics.com/duo-build/mecanum-drivetrain-v2/example-code-and-configuration>

Ex Code (JCR): <https://jc-robotics.github.io/resources/blog/programming-mecanum-wheels>

Wikipedia: https://en.wikipedia.org/wiki/Mecanum_wheel

Programming a TeleOp Mode With Mecanum Wheels:

**Disclaimer: There are multiple correct ways to do this! This is only one example.*



The above OpMode represents the most basic mecanum wheel logic.

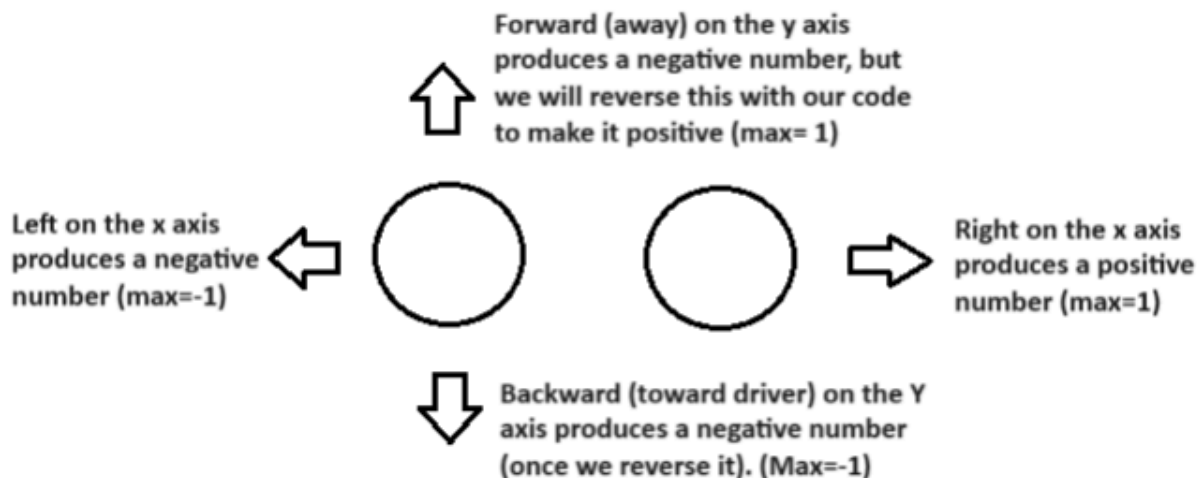
On most chassis, the motors are placed in a symmetrical, mirrored configuration. This means that **either the two left or the two right motors will have to be reversed**. **The right 2 wheels needed to be reversed on the robot that this code was built on.*

In order to find the blocks for the motors, you need to have already configured the motors and named them (*the motors in this example are named frontleft, backleft, frontRight, backRight*). Giving them logical names like this is highly recommended. Once they are configured, the blocks associated with the motors can be found in the drop-down menu under “**Actuators**” “**DcMotors.**” The other blocks here can be found under “**Logic,**” “**Gamepad**” and “**Math.**”

- The above TeleOp mode will go forward and backward when the left joystick is pushed forward and backward in the Y axis (away and toward the driver).
 - A negative symbol has been inserted before the LeftStick Y entry because the joysticks come from the manufacturer configured in such a way that pushing the joystick away from the driver is a negative value and pushing it toward the driver is a positive value. This configuration is not intuitive to most people, so it is advisable to reverse the Y axis in all TeleOp modes.
- It will strafe left and right when the left joystick is pressed left and right.
- It will turn counterclockwise and clockwise around its central axis when the right joystick is pressed left and right.

Configuring a DC Motor:

https://ftc-docs.firstinspires.org/en/latest/hardware_and_software_configuration/configuring/configuring_dc_motor/configuring-dc-motor.html



It can be a little bit confusing to understand the logic behind the mecanum wheels, but let's consider some examples:

Moving forward: If you want it to go forward at full power, you will press the left joystick forward (away from you) to the max (max power=1 because we reversed the value of the Y joystick). In this example, you are giving a power of 1 to the left joystick Y axis and a power of 0 to the left joystick X axis and a power of 0 to the right joystick X axis.

So... looking carefully at the blocks, you will see that all 4 motors will receive a power of 1 (they will add nothing from the other joystick inputs). This means that all 4 motors will all move forward at a power of 1 together to create a forward motion.

Moving to the right (streif): Note: Pushing a joystick all the way to the right gives a power of 1 (left gives a power of -1). In this scenario, the left Y axis and the right x-axis receive a power of 0. So, let's look at all 4 motors. Front left motor will receive $0+1+0=1$. Back left motor will receive $0-1+0=-1$. Front right motor will receive $0-1+0=-1$. Back right will receive $0+1+0=1$. So, the front left and back right get a power of 1 (forward motion, because we reversed the Y axis by adding the negative at the beginning). The back left and front right motors both get a value of -1, which will make them go backward. Adding up the mecanum logic, the robot will strafe to the right.

Joysticks and Triggers:

Certain buttons on the controller are simple- just an on/off switch. However, there are a few buttons that provide more sensitive data (triggers and joysticks). For joysticks, a partial push gives a number less than 1. Now that we have our mecanum logic in place, the variables from our robot will be added up to create dynamic movements at different speeds in any direction.

The problem:

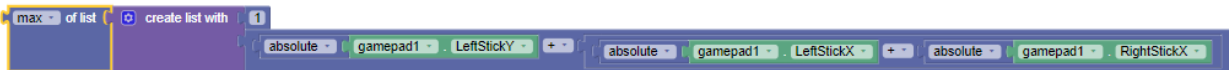
The program shown above WILL work, especially if you go slow and don't try to do too much at once. However, we have accidentally created an error that will become evident once we use more complex movements and faster speeds. Imagine that you are pressing fully forward (away from you) on the left Y axis and fully right on the right x axis. This returns a value of $1+0+1=2$. However, the motor will round that number down to its max speed, which is 1. If the robot is constantly rounding down to 1, the ratios will be off and the robot will move in weird ways.

The Solution: The Denominator (or Regulator)

For every motor, we can divide the total power of each motor by the max power put out by any joystick reading. This will make sure that everything stays proportionate instead of just having some values round down to 1 without changing other values in a relative fashion.



*This is the same code, but with added in at the denominator added to the end... let's zoom in to just the denominator part at the end. Each motor's number is DIVIDED by the following:

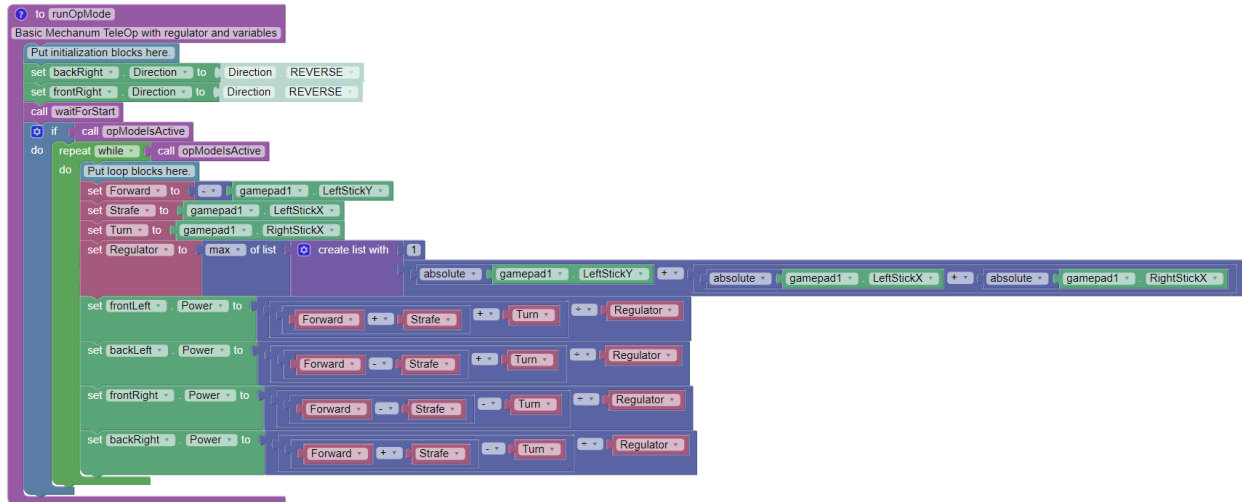


*The blocks are found under the “Logic,” “Math” and “List” drop-down menus. The “create list with” block can be modified with the gear icon (it pre-populates with 3 list items, but we cut it down to 2 items in this case).

Using Variables to streamline:

A variable doesn't mean anything by itself, but we will define what we want it to mean within our code. Using variables will allow us to work much more efficiently.

Here is an example of how we can use variables to streamline our mecanum TeleOp Mode:



To create a variable, we go to the “**Variable**” drop-down menu and select “**create variable.**” It will then pre-populate some useful blocks for us that refer to our new variable.

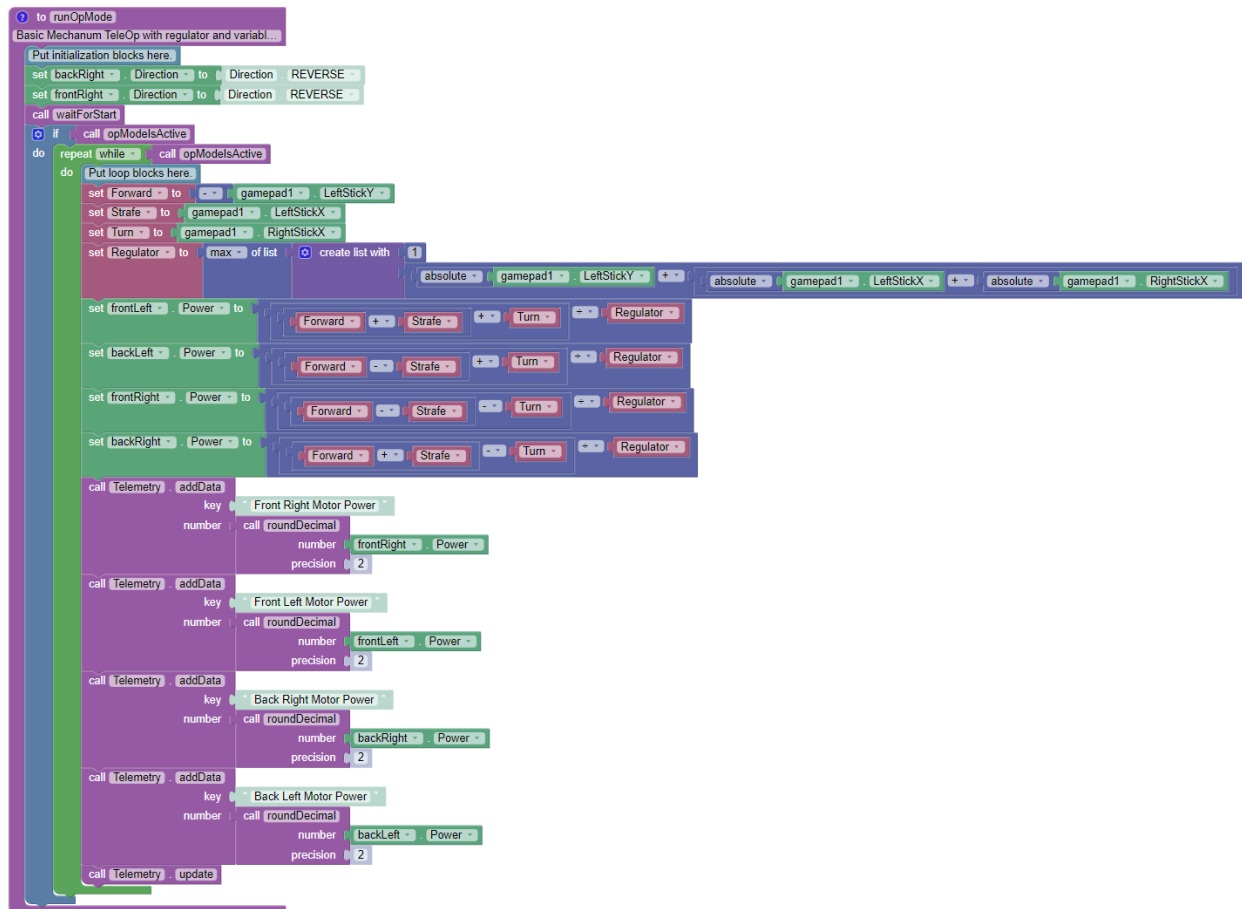
In the above example, we have introduced the variables “**Forward**” “**Strafe**” “**Turn**” and “**Regulator.**” *The “Regulator” is the same as “The Denominator” in the previous section.

We set “**Forward,**” “**Strafe**” and “**Turn**” variables to the joysticks. Notice that we placed the negative value in front of the “**gamepad1.LeftStickY**” block. So, now our Y axis is already inverted and we don't have to put the negative symbol before our forward instruction in the logic associated with each motor power.

In our motor power section, we have used the variables that correspond to our joystick motions to fill out our mecanum logic.

We also set our “Regulator” variable to define the max value we will be dividing all the motor power by to create smooth motion.

Adding Telemetry:



It might be useful to display telemetry data. In the above example, telemetry data is being displayed on the driver station for all 4 motors. The **“key”** is what the power is labeled as on the driver station. The **“number”** is linked to the power of each motor (so it will return a number between -1 to 1). This type of telemetry block allows us to limit the number of decimals in the number that is returned by entering in the **“precision”** that we want returned (*limited to 2 decimals in this case*).

*Don't forget that **“callTelemetry.update”** block at the end.

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 mecanum wheels installed correctly? If they are not in the correct orientation, they will not work.
- 2) 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?
- 3) Is there any physical object stopping my robot from moving? This could be an internal component, like something stuck in the gear, etc?
- 4) Do I have all of my wheels set to "forward" or "reverse" correctly?
- 5) Did I select all 4 different motors in my "set power to" section and in my "regulator/denominator" section?
- 6) Did I double check the + and - symbols on my mecanum wheel logic?
- 7) Did I remember to multiply the Y-axis variable by "-" when I set my "forward" variable? Gamepad Y-axis is reversed from what I want.
- 8) Are all of my instructions inside the green "**repeat while.call.
OpModeisActive.do**" box?
- 9) No telemetry? Did I remember to add the "**call.telemetry.update**" box?
- 10) Did I select the correct TeleOp from my drop-down menu on my driver station?
- 11) Is my robot's battery running low?