Exponential vs. Linear Control

This lesson will build upon lessons 16 and 17 by further enhancing your arcadeDrive method. You'll change it from using a linear mapping of the joystick value to the motor outputs, to an exponential one, and compare how the drive characteristics feel with each control scheme.

A *linear* mapping of the joystick values is straightforward – a .25 value on the joystick translates to .25 motor output, .5 to .5, and so forth. This makes sense, but it isn't necessarily the best way to control a robot. Often, when the driver wants to go fast, they want to go *fast* – they're probably going to apply full power to the robot. It's often rare that a driver will want to drive at, say, 80% power. However, when they want to go slowly, they might want to go at different slow speeds. For example, they might want to barely move, or cruise but slowly, or move just slow enough to maintain full turning control. Example desired motor outputs in these cases might be .25, .33, or .5. However, for a human driver, it can be difficult to physically create the difference these values on a joystick, since the distance between .25 and .33 on a joystick is fairly small.

With an *exponential* mapping, the input from the joystick is raised to some power, typically two (or in other words, squared) before being sent to the motors. So a value of .1 would become .01, .5 would become .25, a value of .75 would become .5625, and a value of 1 would stay 1. The fact that 1 still corresponds to 1 is important, so that you don't lose any of your maximum output range.

When you square the inputs, a much larger portion of the physical joystick range corresponds to the smaller range of possible outputs. For example, with linear control, the physical range between 0 and .5 is, of course, 50% of the joystick's travel distance in a single direction. (Ignore deadbanding for this example.) With exponential control, a value of .707 corresponds to 50% (.707^2 = .5), so you have almost 71% of the joystick's travel distance covering the same range.

Exponential control is not necessarily "better" than linear control, but many drivers do feel that it improves robot handling when they try it. The Romi is fairly small, low speed, and easy to control either way, but you can make the decision for yourself after you try both out. The Romi actually uses exponential drive by default but you can change this by changing the "true" value on line 167 of DifferentialDrive.class (accessible by F12'ing on the "DifferentialDrive" class found in the class fields of Drivetrain.java) to false, as this variable corresponds to a the boolean parameter "squareInputs" in the arcadeDrive method declared on line 179 of DifferentialDrive.class and does exactly what is described in the prior section. Or, you can skip that, implement it yourself, and try it out after doing so, because regardless of which one you prefer you'll implement both in this project.

The arcadeDrive method declared on line 179 of DifferentialDrive.class has all the logic that makes the robot drive and turn, so you can use that as a reference as you complete this project. Note that there are multiple arcadeDrive methods in this file because of overloading, but the one declared on line 179 is where the important logic is. Lines 204 through 222 handle combining the rotation value with the speed value to create the net values sent to the motors, and lines 224-226 handle setting the motors. (Line 225 multiplies one side by -1 if necessary, because one motor is functionally driving in reverse, because the motors are physically pointed in different directions on the robot.)

Doing the Project

For your project, implement your own arcadeDriveCustom method in Drivetrain.java. You'll need to copy some of the code mentioned above unless you want to do the math yourself. In theory you

could copy the whole method if you want, but even if you do this you'll still have some work to do because this method exists within the DifferentialDrive class and you're going to make your method in Drivetrain.java. To start, comment out this line in Drivetrain.java:

```
// Set up the differential drive controller
private final DifferentialDrive m_diffDrive = new DifferentialDrive(m_leftMotor, m_rightMotor);
```

This will give you a syntax error because you won't have an m_diffDrive object anymore:

```
m_diffDrive.arcadeDrive(xaxisSpeed, zaxisRotate);
```

Instead of calling the arcadeDrive method of the DifferentialDrive class, you'll replace the line that now gives a syntax error with a call to your newly created method. The goal will be to do that and have your robot still drive. Here's what your new method call should look like. Of course, if you type this in right away you'll get a syntax error, but the project is to create this method and make it drive your robot.

```
arcadeDriveCustom(xaxisSpeed, zaxisRotate);
```

A Few Pointers

The hardest part is getting started so here are a few pointers to help you out.

- You can start by commenting out the lines as shown above and replacing them with the arcadeDriveCustom method call. As soon as you define that method, your syntax errors will go away, but your robot won't drive anymore until you fill in your new method with code.
- The Spark objects declared in Drivetrain.java have "set(double speed)" methods that will send output to the motors, so for example you could use the following lines to set them to full power:

  ```
  m_leftMotor.set(1);
  m_rightMotor.set(1);
  ```

- Since you need to implement both linear and exponential control, it's probably easiest to start with linear control and get your robot moving before tackling exponential control.
- An easy test to confirm that you're reading in your input the right way, and setting output the right way, is setting the left motor to the speed value and the right motor to the turn value. This won't be "arcade drive" anymore, it will be "tank drive", so feel free to try driving your robot like that. It will work well on an Xbox controller but not a single joystick. It will still *work* on a joystick, it just won't be comfortable to drive. Also, the variable names "xaxisSpeed" and "zaxisRotate" would become incorrect, as the values would represent "leftSpeed" and "rightSpeed'. But, it will confirm for you that you're properly getting the joystick values and outputting them to the motors. You don't have to do this, it's just an option for getting starting.
- As stated, you can copy most or all of the arcadeDrive method from DifferentialDrive if you want. But you don't need all of it to complete the project, as that method has a lot of additional stuff. All you need to do is do the math to create linear and exponential drives. As mentioned before, lines 204-222 are the lines that combine the speed and rotation variables into left and right motor output values, which you would then use in your m_left and m_rightMotor.set() methods.
- You don't need to worry about any of the code from the DriveStraight project in this project, although you could implement that if you want.

- When you square inputs for the exponential control mode, make sure you don't turn negative numbers into positive numbers. Remember, 1 squared equals one, but -1 squared also equals one. There are a couple different ways to account for this. You can use if statements to multiple negative values by -1 after you square them, but a better way is demonstrated on lines 195 and 196 of DifferentialClass.java.