Reminder: from lesson 16 onwards, the projects are aimed for you to do with less guidance, fewer or no direct steps, and without solution projects provided.

In lesson 9, we briefly talked about the concept of *deadbanding*, or creating a zone of input from the joysticks which register as zero, even if the input is not literally zero. You may recall that this can be useful because the joysticks don't physically return to a perfect zero and might register some value like .05 instead when at rest. If you don't remember much about deadbanding, you'll want to review lesson 9. In this project, you'll add code to your arcadeDrive method from lesson 16 to create a deadband on the user input. You'll also scale the rotation value based on the magnitude of the speed value. That is similar to what you did when you implemented cut power mode, except instead of a boolean value of either cutting the power or not, you'll scale the turning power dynamically based on the speed. For example, if the speed is 0, you would not want to scale down the turning at all, but if the speed was 1, you might want to scale it down by some amount, say 50%. In that case, you'd be scaling down the turning power by .5 multiplied by the speed value. So a speed value of 1 would be a 50% reduction in turning power, a speed value of .8 would be a 40% reduction in turning power, a speed value of .2 would be a 10% reduction in turning power, etc.

Deadbanding

To complete the deadbanding portion of this project, create a method in Drivetrain.java called "applyDeadband" that accepts a double representing the input value, and returns a double that is the input value after deadbanding is applied. Create a value in Constants.java that specifies what the minimum threshold for input from the joysticks needs to be in order to pass the deadband filter. A decent first guess might be .1, but you'll have to determine this for yourself based on the joysticks you're working with. Then, your method simply has to check if the magnitude of the input exceeds the deadband threshold, or not. If it does, you can simply return the input. If it does not, you return 0. Make sure your applyDeadband method returns the input with the same *sign* that was passed into it – that is, if a positive value was passed in, a positive value (or 0) should be returned, and if a negative value was passed in, a negative value (or 0) should be returned. When your method is done, go back to your arcadeDrive method and pass your input values to your new method, and update their values to whatever the method returns. That's a mouthful but this is very simple, because you can update the value of a variable, and pass that variable into a method, all in one line of code. Your lines that call applyDeadband should look like this:

```
xaxisSpeed = applyDeadband(xaxisSpeed);
zaxisRotate = applyDeadband(zaxisRotate);
```

Turn Scaling

To complete the turn scaling portion of this project, create a method in Drivetrain.java called applyTurnScaling that takes two doubles, one for speed, and one for rotation, and returns a double that is the scaled turning value. You can update your zaxisRotate value in arcadeDrive the same way you updated it for the deadbanding portion, except by calling applyTurnScaling in your drive method. Inside the applyTurnScaling method, you need to figure out how much to reduce the turning, which you can do by multiplying the speed value by a new Constants.java value you'll create that represents how much you want to scale the turning down, for example .5 if you want to scale the turning down by 50% at max speed as explained in the example at the start of this lesson. This represents the percent you want to reduce the turn power by – for example, if your speed is .8 and you're using .5 as your constant, you'd want to scale down the turning by 40%.

Once you've determined your percentage reduction in turning, you can subtract that number from 1 to create a turn scaling coefficient. For example – let's again say the speed is .8 and the scale factor is .5. Then, .8 * .5 = .4, which is your turn reduction percentage. 1 - .4 = .6, so .6 is your turn scaling coefficient. The last step is to multiply your turn input value by your coefficient to determine the final turning value. Here's a full example, written out in English and shown with numbers:

1. Multiply by the speed by the Constants.java turn scaling value, and save this value to a variable called "turnReductionPercent". For the example, assume the speed is .8 and the turn scaling value is .5.
    a. .8 * .5 = .4
2. Subtract the turnReductionPercent variable from 1, and save this to a variable called "turnCoefficient".
    a. 1 - .4 = .6
3. Scale (multiply) the original turn value by the turnCoefficient. Assume the original turn value is 1.
    a. 1 * .6 = .6
4. Return the final result, in this case .6.

Testing

Once you've created your two methods and utilized them in your arcadeDrive method, make sure to test them by driving the Romi around. The deadband method can be tested by trying to drive your robot while looking closely at the Robot Simulation interface. Pick the Romi up and hold it in your hand so there is no friction on the wheels, and slowly start pressing on the joystick while looking at the joystick values on the interface. The wheels should start spinning right as you cross the deadband threshold.

To test turn scaling, put the Romi back down and start spinning it in circles without giving it any forward/backward speed. Then start giving it forward/backward speed. You should see the rate of turning decline as you give it more speed. You can increase the value of the turn scaling constant in Constants.java to see a more drastic effect, which is useful for testing.