Radius Turn Theory and Implementation

Nick Schatz, Team 3184 Blaze Robotics

December 12, 2016

Abstract

This document outlines the theory and implementation in Python of constant-radius turning for FRC robots. It begins with a section about constant-radius turning and why one would use it. The second section explains the theory of constant-radius turning for a tank-drive (skid-steer) robot. The final section includes code used for implementing constant-radius turning and discusses modifications.

1 Introduction

Traditionally, skid-steer robots are driven using the method of either tank drive or arcade drive (or some variation). With tank drive, the driver has direct control over the power of each drive train. This makes the robot difficult to control precisely and hard to drive straight. Arcade drive solves these problems by controlling the forward motion of the drive train on one axis, and moving the turning power onto another (perpendicular) axis. However, arcade drive comes with another problem. The turning radius of the robot is dependent on the applied forward power, not just on the applied turn power. Constant-radius turning solves this problem by making the turning radius of the robot depend only on the applied turning power, completely independent of the applied forward power. This results in a more natural control scheme, easier to learn and drive.

2 Theory

Consider an idealized skid-steer robot with track width D. Let the robot make a turn where the robot makes an arc of radius R with its center.

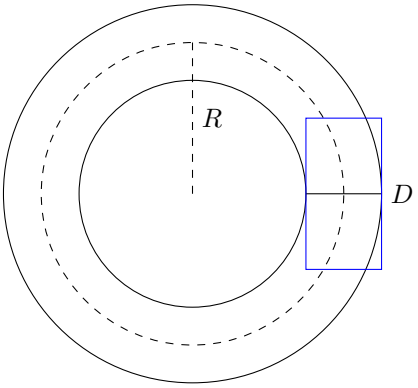

Figure 1: A skid-steer robot with track width D, to travel along on arc of radius R

The distance the outer wheel travels we will define as X_O, and the inner distance as X_I. Therefore,

$$X_O = 2\pi(R + D/2) \tag{1}$$

and

$$X_I = 2\pi(R - D/2) \tag{2}$$

At this point, we will begin to treat wheel power (or speed) and distance over a set period as equivalent, and replace X with V.

If one wished to drive a robot along a set arc for a set period of time, they might look no further. However, we wish to take input from a driver in the form of speed and turning radius. Let the desired speed be S and the radius be R, as it has been. We must relate all four variables: V_O, V_I, S, and R. The key is that the ratio between the outer and inner wheel speeds must be the same for some radius at any speed. First, find the ratio between the outer (1) and inner (2) speeds:

$$\frac{V_O}{V_I} = \frac{R + D/2}{R - D/2} \tag{3}$$

It is clear that V_O should always be greater than V_I. Because the robot has an upper limit on speed, we will set $V_O = S$. and solve for V_I:

$$V_I = S\frac{R - D/2}{R + D/2} \tag{4}$$

This is the fundamental equation for constant-radius turning, which we will be implementing in the next section.

3 Implementation

3.1 Choosing a radius

A straight line is an arc of infinite radius (or as the radius approaches infinity, the curve approximates a straight line.) This is not very practical from a programming perspective. Therefore, a special case is needed for when the driver wishes to drive in a straight line.

Also as a consequence of the maximum radius being infinite, we need to choose a finite maximum radius to turn as a percentage of. This choice is relatively arbitrary, but should be sufficiently large to allow very slight turns at humanly possible joystick angles. This should be on the scale of 200 feet, but is an option for tuning. [1]

Since it is natural that a farther joystick turn should be a wider radius, the turn radius should be $R = M * (1 - |x|)$, if M is the maximum radius and x is the inputted turning power, on a range of [-1, 1]. However, this always produces a positive radius. Correcting the sign would give the correct ratios, but since we have fixed V_O at S, it may produce a speed too high for a wheel to turn at. The solution is to apply power to the wheels based on which we determine is the outside and inside. If $x > 0$, then the robot is turning right and the left wheel will be the outer wheel, and vice-versa.

3.2 WPILib Implementation

A function to turn at a constant radius exists in WPILib as RobotDrive.drive. It takes an output magnitude and curve value, where the curve value is $e^{\frac{-R}{D}}$. Although the documentation notes that it would likely be used in autonomous routines, it is also suitable for driver control using the radius-choosing method described above.

[1] Until this point, units have been ignored. They do not matter, as long as all units agree.

3.3 Custom Implementation

Below is a function in Python with robotpy to drive at a power along an arc.

```
def radius_turn(robot_drive, power, turn, robot_width, max_radius):
    """
    Turns at a constant turn radius at a given power
    :param robot_drive: wpilib.RobotDrive object
    :param power: The power to drive at. [-1, 1]
    :param turn: The joystick turn value. [-1, 1]
    :param robot_width: The track width of the robot
    :param max_radius: The maximum turn radius of the robot
    """
    D = robot_width
    if turn == 0:  # Special case to handle straight lines
        robot_drive.tankDrive(power, power)
        return
    # Calculate the turn radius
    radius = max_radius * (1 - abs(turn))
    Vo = power
    # Calculate the inner wheel power
    Vi = Vo * (radius - D/2) / (radius + D/2)

    # Apply the calculated power to the correct wheels
    if turn > 0:
        robot_drive.setLeftRightMotorOutputs(Vo, Vi)
    else:
        robot_drive.setLeftRightMotorOutputs(Vi, Vo)
```

3.4 Feedback Control

This same method can be used for closed-loop driving by inputting power as a desired speed of the robot (in real units, such as feet/second) and outputting similar values into two control loops, one for each side of the drivetrain. Because each drivetrain side is calculated as a ratio of the other, this requires no further modification to the function.

3.5 Further Considerations

This code may cause the robot to feel unresponsive at low joystick values, and twitchy at high values. This is because of the nonlinearity of equation (4). For smoother operation, taking some root of the joystick turn value may be desirable (say, $\sqrt[3]{x}$)

A joystick value of +/-1 will cause the robot to turn with radius zero, at a speed of the inputted power. This may not be a desirable operation, but can be avoided by maintaining a minimum radius, possibly as half the robot's track width. If so, an alternate mode that allows turning at a zero radius may be helpful, possibly by adding a special case when no drive magnitude is being inputted.