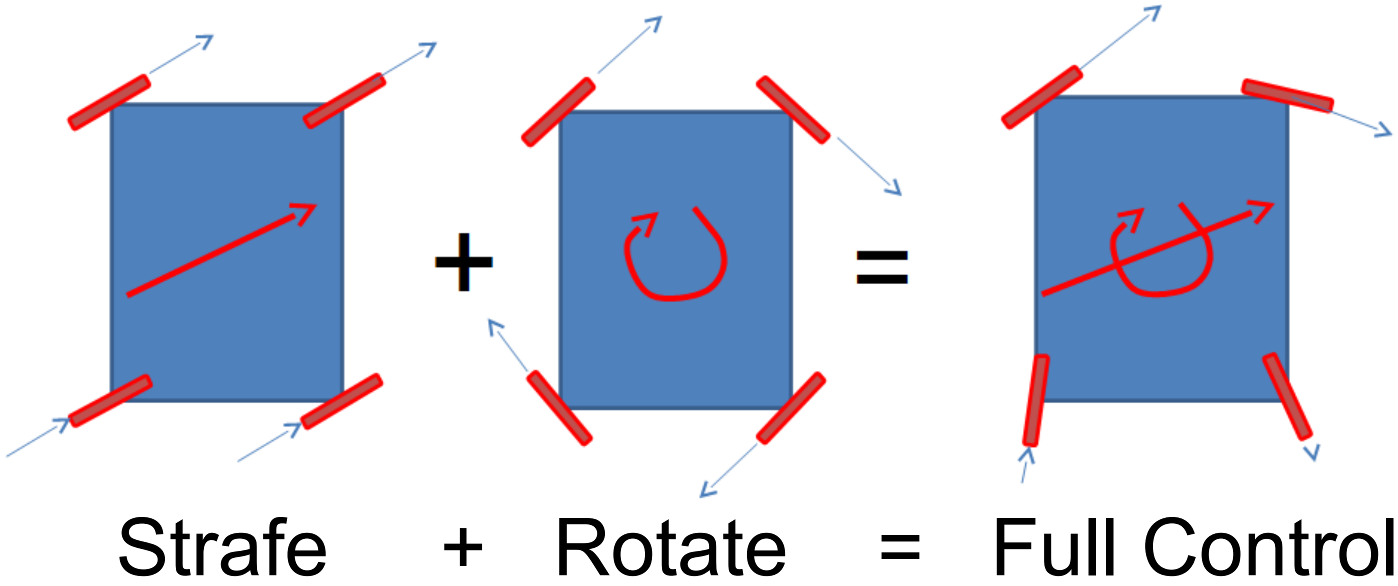
Swerve Drive

FRC Team 2067, Apple Pi

Swerve Drive: 4 Independent Wheels



Each wheel will be driven and steered independently.
Requires 4 drive motors (CIM) and 4 steering motors (AM-9015).

Strafing

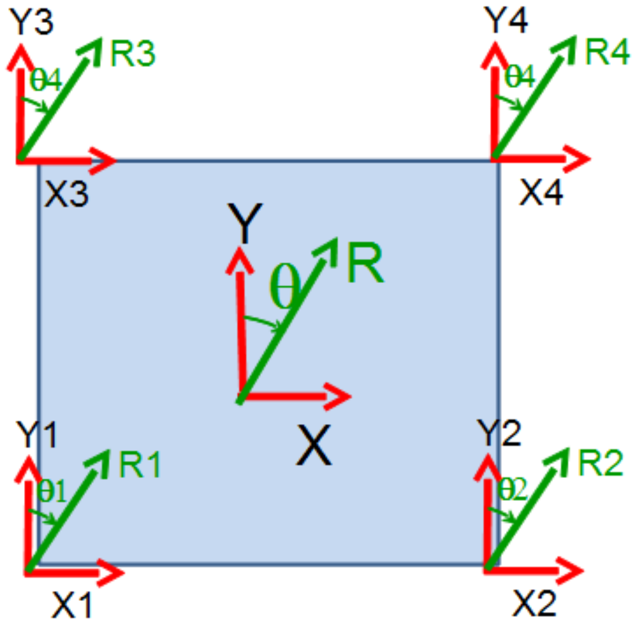
X and y are the joystick inputs for strafing/translation.

Each wheel has the same angle and speed.

Take the x and y values and convert to the wheel angle and speed. Note: you must first negate the y value, because forward is negative on joysticks.

r = wheel speed = $\mathrm{sqrt}(x^2 + y^2)$

θ = wheel angle = atan2(x, y)

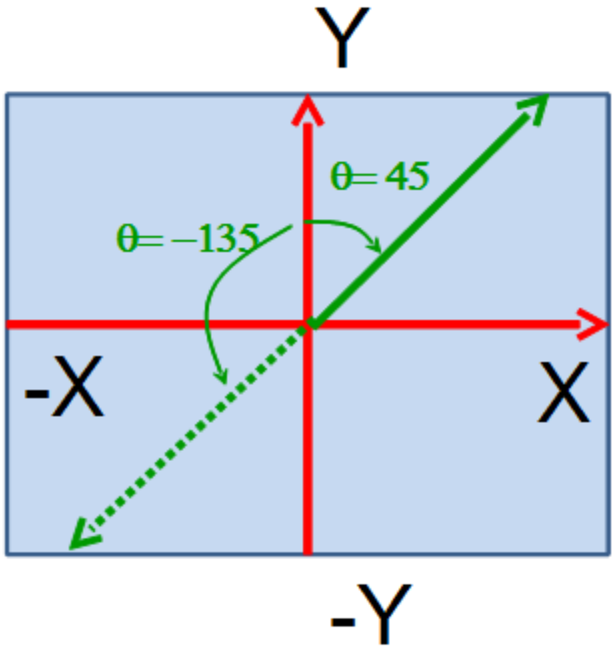


Caution with Arctangent

We use atan2(x, y) instead of the normal arctan(x / y) to return the correct quadrant.

Taking arctan(x / y) isn't exactly correct:
For x = 1 and y = 1, arctan(1 / 1) = 45
For x = -1 and y = -1, arctan(-1 / -1) = arctan(1 / 1) = 45, though you want -135.

To fix this, programming languages have atan2.
This takes x and y separately and adjust arctan to the right quadrant.

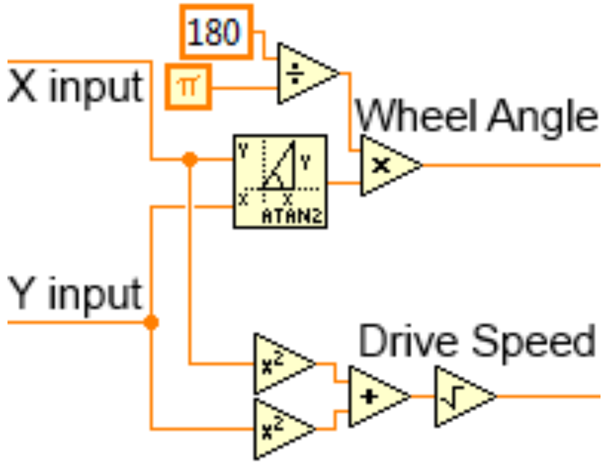


Strafing: LabVIEW Implementation

You only need to calculate this once for all the wheels, as they will all be pointing the same direction and moving at the same speed.

LabVIEW's atan2 function computes atan2(y, x) by default, but we want atan2(x, y). To fix this, switch x and y on the block.

LabVIEW also does trigonometric functions in radians, so you must convert it to degrees.



Remember:

$\theta_{degress} = 180/\pi * \theta_{radians}$

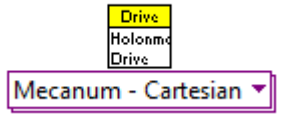
Strafing: Field-Centric Control

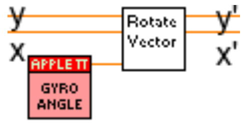
A useful modification can be added to the strafing code to make it field-centric.

This means that when the driver pushes up forward on the joystick, the robot moves forward relative to the field, regardless of what angle the robot is pointing at.

To do this, we take the (x, y) joystick input and rotate it by the gyro reading. The gyro gives the angle that the robot is facing.

You can borrow the Rotate Vector block from inside the cartesian mecanum block:



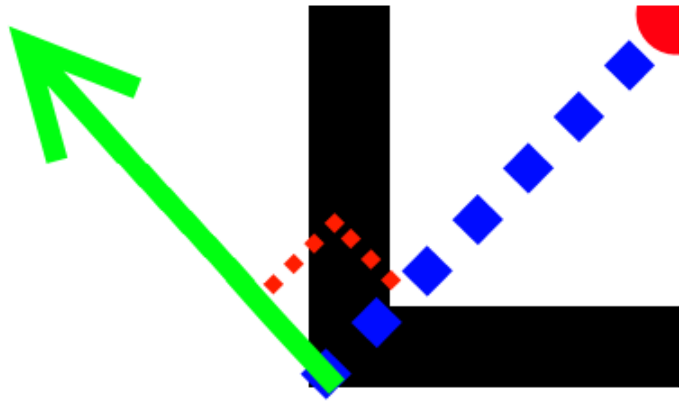


Once the inputs are rotated, put them through the same calculations shown on the last slide.

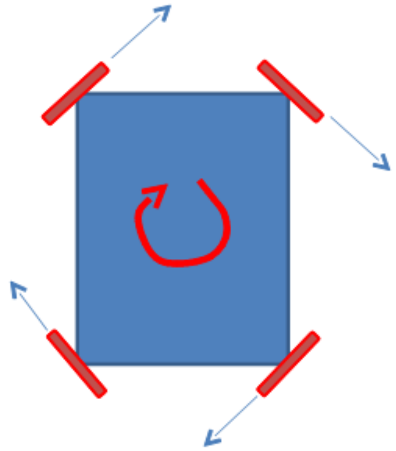
Rotation: Overview

Unlike strafing, each wheel has a different angle

To get the angle and the speed for each wheel, we find a vector pointing from the wheel to the center, then rotate it -90 degrees.



These perpendicular vectors create rotation:



Rotation: Point to Center

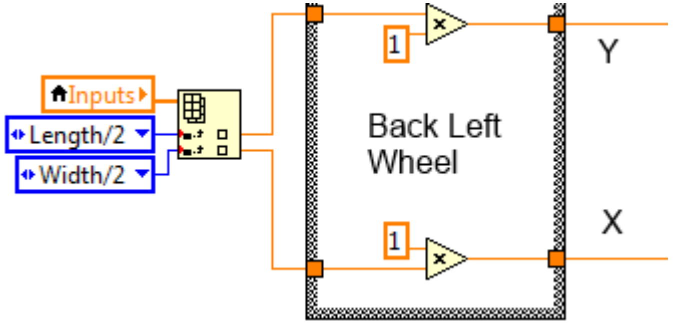
First, find the vector from each wheel pointing to the center of the robot.

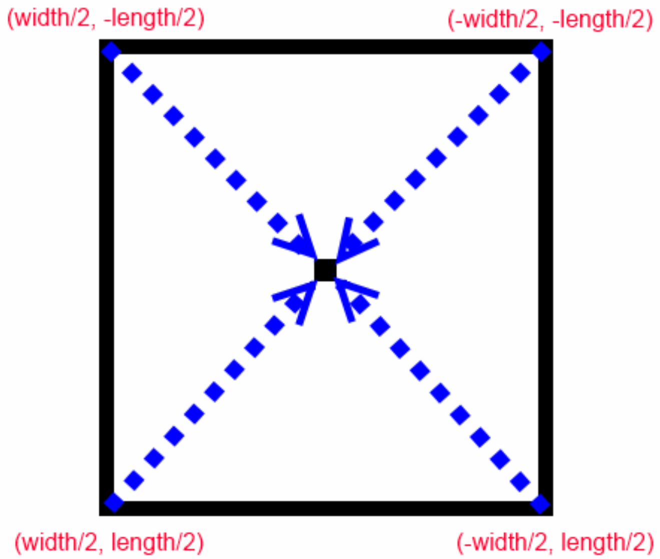
These (x, y) vectors are essentially just the distance from each wheel to the center of the robot.

FL: (width/2, -length/2)   FR: (-width/2, -length/2)

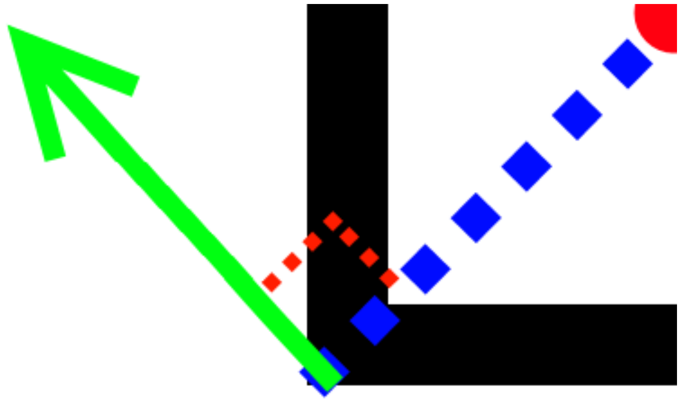
BL: (width/2, length/2)   BR: (-width/2, length/2)

Here is the LabVIEW code for the BL wheel:





Rotation: Perpendicular Vector

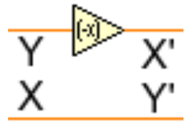


To find the perpendicular vector, the one pointing to the center of the robot must be rotated -90 degrees.

A shortcut for rotating a vector -90 degrees is by setting x to y, and -y to x.

$x_{new} = -y_{old}$
$y_{new} = x_{old}$

In LabVIEW:



Rotation: Scaling the Vector

At this point, you would have a (-length/2, width/2) vector for the back left wheel.

However, there is a problem with this: it needs to be scaled down before you can use it. Remember the formula for finding the wheel speed from an (x, y) vector:

r = wheel speed = $\text{sqrt}(x^2 + y^2)$

If you tried to plug this vector into the equation for a 30x30 robot, the BL wheel speed would equal 21.2.

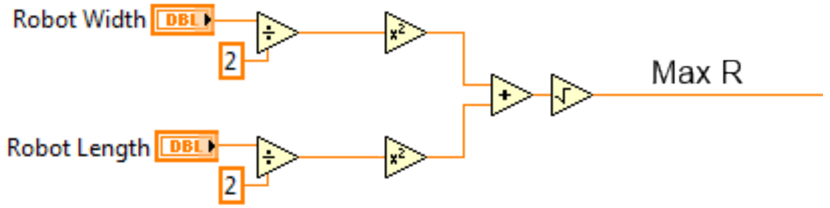
Valid wheel speeds are in the range -1 to 1, so this obviously wouldn’t work.

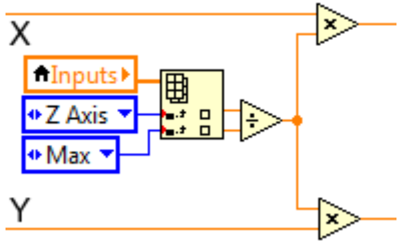
Rotation: Scaling the Vector

To scale it down, we must divide x and y by the maximum r of the wheels, so that the new values of r will be no greater than 1.

This maximum value is sqrt((length/2)^2 + (width/2)^2).

Once this is done, you must scale it again according to the user's z axis input in order to rotate at the desired speed. This is done simply by multiplying the vector by the z axis of the controller.

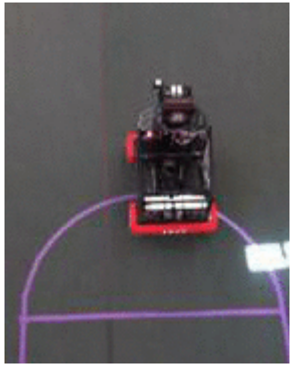


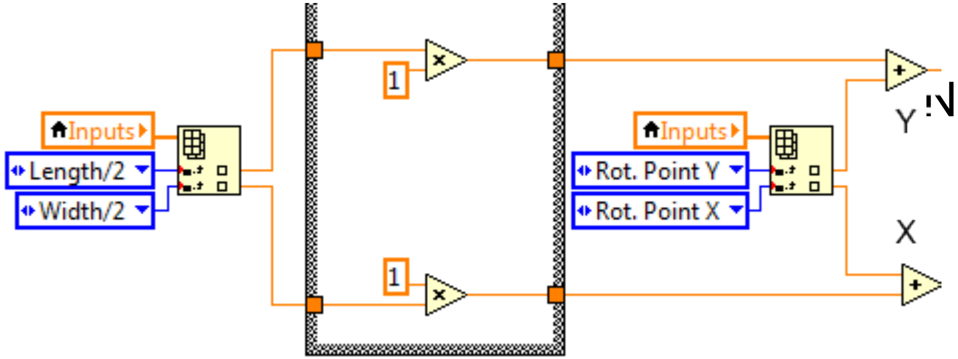


Rotation: Rotate Around Any Point

This code can easily be modified to rotate around any point, not just the center of the robot. To do this, a few changes must be added.

Point the initial vectors to the rotation point, not necessarily the center of the robot. Here is an example for the back left wheel.


1717 rotating around a wheel



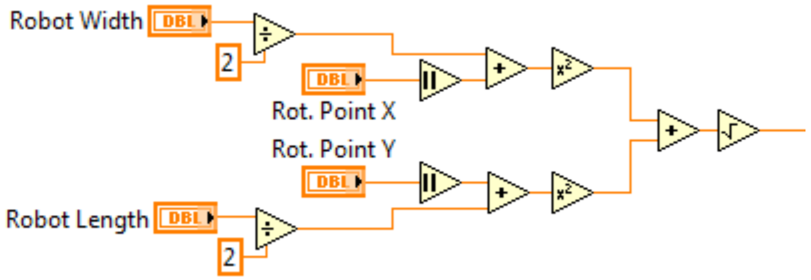
Note:

The rotation point is relative to the center of the robot.

Rotation: Rotate Around Any Point

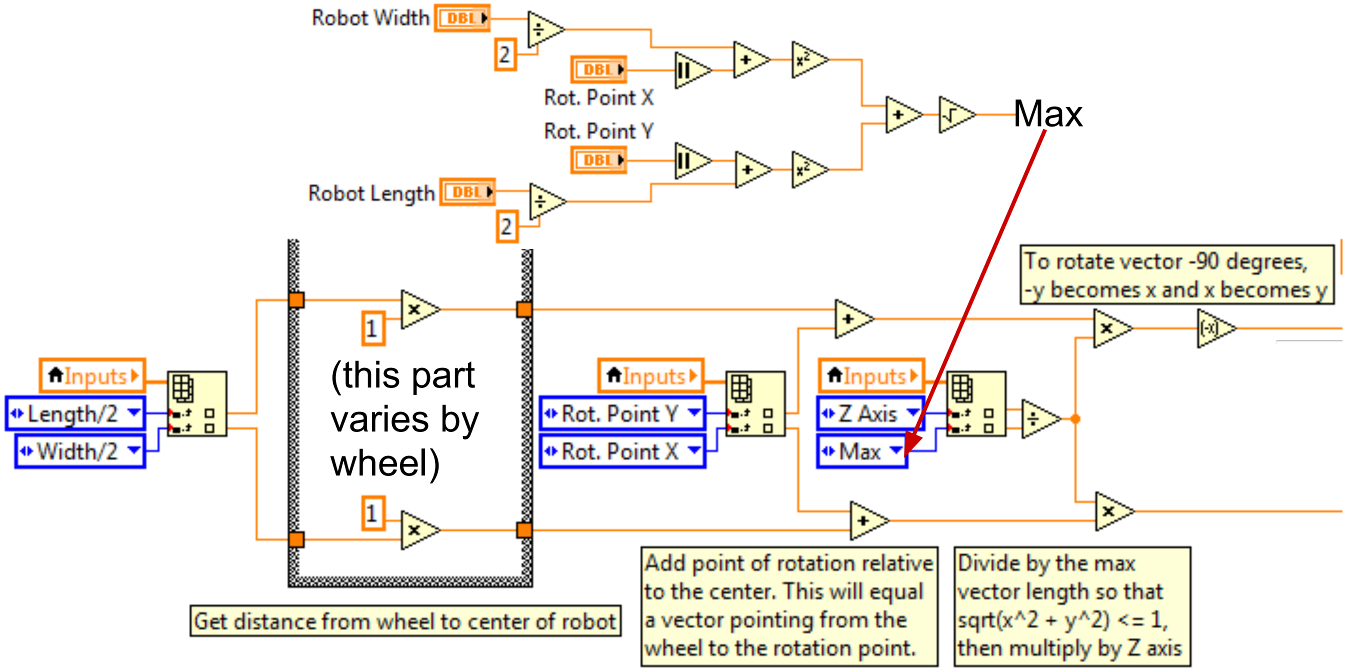
The other change is, when scaling the vectors, the maximum r is not necessarily sqrt((length/2)^2 + (width/2)^2).

With a custom rotation point, the maximum r can be calculated directly as:
sqrt((width/2 + |rot. point x|)^2 + (length/2 + |rot. point y|)^2)



Rotation: Putting It All Together

Here is a sample from Swerve v3 to calculate rotation for the back left wheel:


Robot Width
DBL
2
Rot. Point X
Rot. Point Y
Robot Length
2
Max
To rotate vector -90 degrees, -y becomes x and x becomes y
Inputs
Length/2
Width/2
1
(this part varies by wheel)
1
Inputs
Rot. Point Y
Rot. Point X
Inputs
Z Axis
Max
Get distance from wheel to center of robot
Add point of rotation relative to the center. This will equal a vector pointing from the wheel to the rotation point.
Divide by the max vector length so that sqrt(x^2 + y^2) <= 1, then multiply by Z axis

Vector Addition

Once you have separately calculated the vectors for strafing and rotation, you can put them together to get full swerve control.

This is done simply by adding the two vectors together.

$(x_{net}, y_{net}) = (x_{strafe} + x_{rotate}, y_{strafe} + y_{rotate})$

Note that these vectors must be in the (x, y) form, not the wheel speed/wheel angle form.

Vector Addition

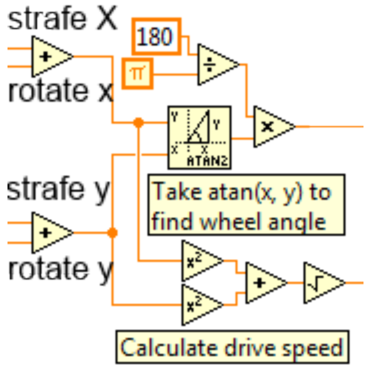
Once you have added the vectors, then convert it to the angle/speed form.

r = wheel speed = sqrt($x^2 + y^2$)

θ = wheel angle = atan2(x, y)

There is one more step before these values can be used.

Once you add together the two vectors, the r value may become greater than 1, so they must be scaled down.

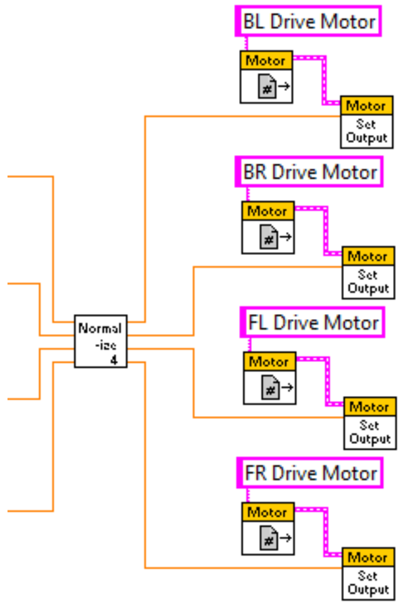


Finalizing The Drive Speeds

There is already a block that can scale down these r values: the Normalize 4 block, found in the Mecanum - Cartesian code where we found the rotate vector block.

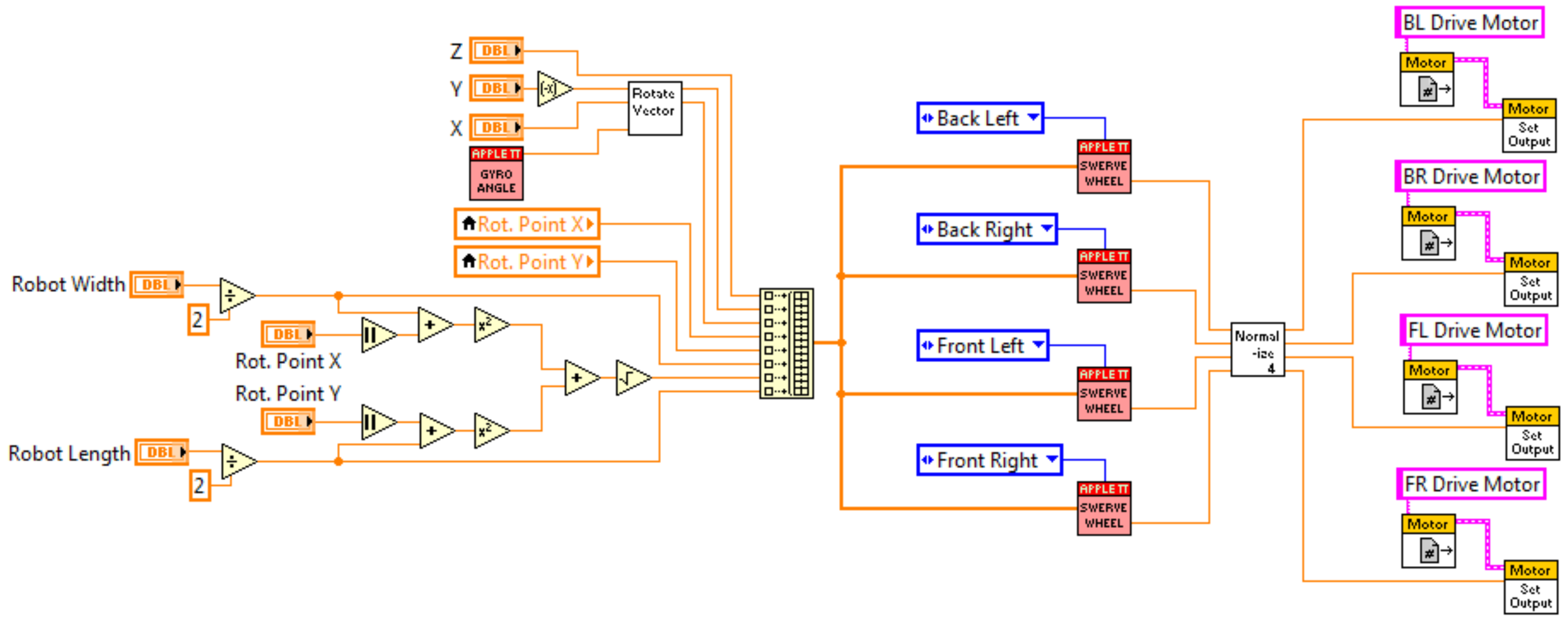
Attach the r values from each wheel into the normalize block, and it will scale them down appropriately if any of them are greater than 1.

Now these r values are ready to be sent out to the motors as the drive speed.



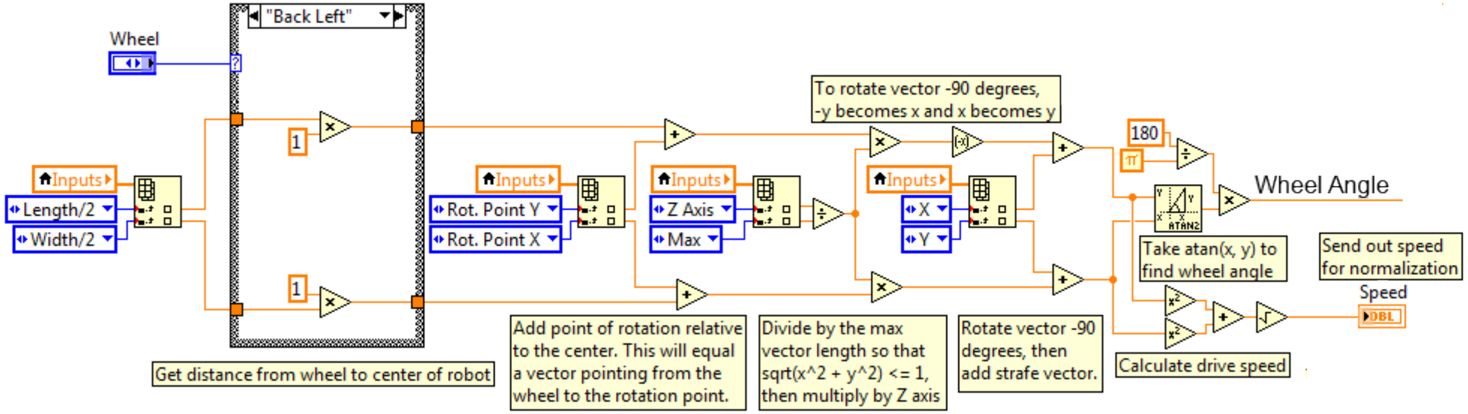
The Code So Far

Swerve Drive.vi



The Code So Far

Swerve Wheel.vi



Get distance from wheel to center of robot

Add point of rotation relative to the center. This will equal a vector pointing from the wheel to the rotation point.

Divide by the max vector length so that sqrt(x^2 + y^2) <= 1, then multiply by Z axis

To rotate vector -90 degrees, -y becomes x and x becomes y

Rotate vector -90 degrees, then add strafe vector.

Take atan(x, y) to find wheel angle

Calculate drive speed

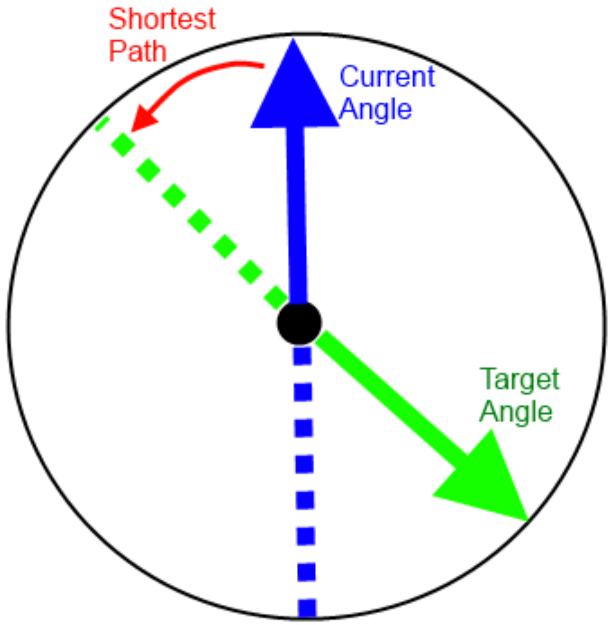
Send out speed for normalization

Processing The Wheel Angle

By this point, we have calculated the target wheel angle, but we still need to tell the wheel how to reach that angle, taking the shortest path to get there.

Consider the following:

The wheel is currently pointing at 0 degrees, and your target angle is 135 degrees. You could turn clockwise until you reached 135, but the shortest path is to turn the other way until you reached -45, and then drive the wheels backward.

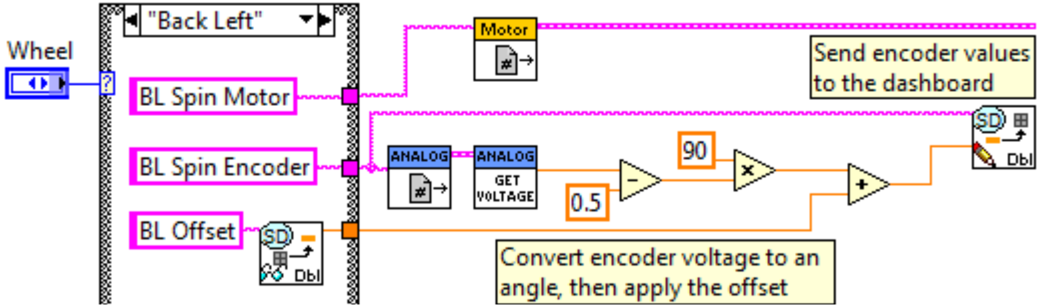


Reading the Encoders

Before we can do these calculations, we need to find out what angle the wheels are currently pointing at.

Read the encoders using the Analog Get Voltage block. To convert this voltage to an angle in degrees, subtract 0.5, then multiply by 90.

Then, you must add the wheel offsets from the dashboard which are necessary to make sure all the wheels are aligned. Also, it is a good idea to send the encoder angle out to the dashboard.

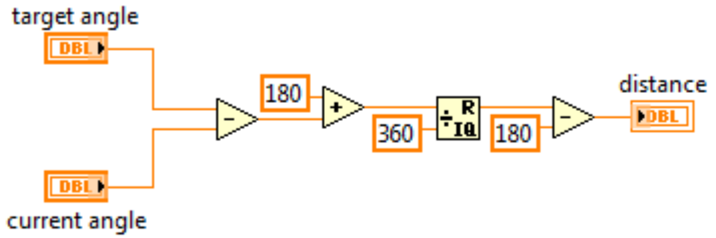


Send encoder values to the dashboard

Convert encoder voltage to an angle, then apply the offset

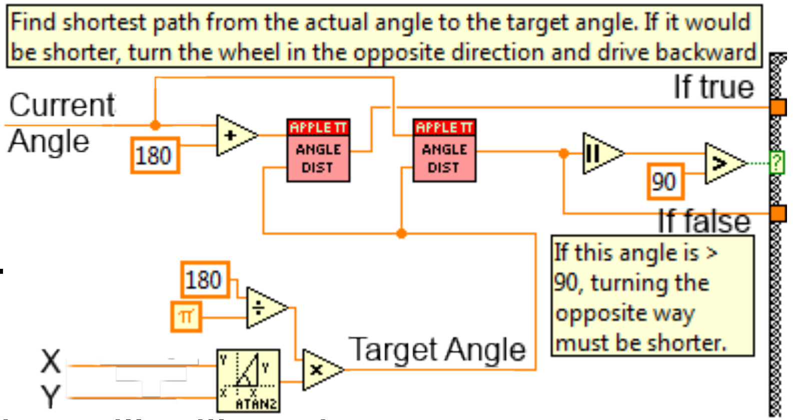
Finding The Shortest Path

When determining the shortest path, we first find the distance between the current angle and the target angle, which will be in the range -180 to 180. We are not considering the “turn the opposite way and reverse” strategy just yet. There are several ways to do this, but here is the method that we use. The “R IQ” block tries to divide the angle by 360 degrees, and gets the remainder. For example, if you inputted 382, the result would be 22. Notice that 382 and 22 are essentially the same angles. The output from this block will always be the same angle, constrained to the range 0 to 360. We add 180 beforehand, and then subtract 180 after, effectively shifting this range to be -180 to 180.



Finding The Shortest Path

To find the shortest angle, we actually need to make two of those angle distance calculations: one between the current angle and the target, and one with the opposite of the current angle and the target. Finding the opposite of the angle can be done simply by adding 180 degrees. It is fine if the angle becomes >360; the calculation will still work.

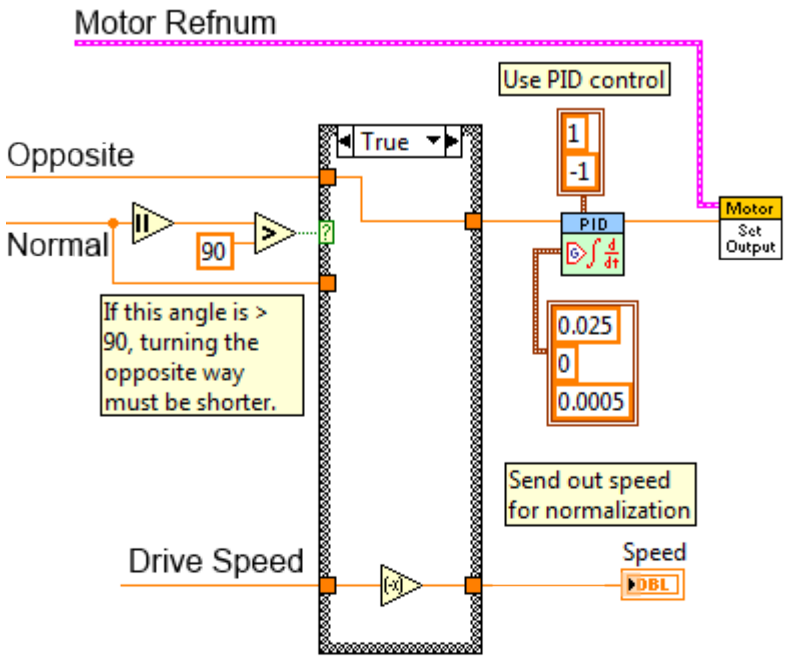

Find shortest path from the actual angle to the target angle. If it would be shorter, turn the wheel in the opposite direction and drive backward
Current Angle
180
APPLE π ANGLE DIST
APPLE π ANGLE DIST
If true
90
If false
If this angle is > 90, turning the opposite way must be shorter.
180
π
Target Angle
X
Y
ATAN2

If the absolute value of the distance between the current angle and the target angle > 90 degrees, it must be quicker to turn the opposite way and drive backward.

Turning to the Angle

Now that we know which path is shortest, we can plug that distance as the setpoint of a PID control.

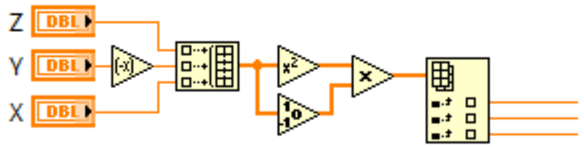
If you are using the opposite angle, be sure to negate the drive speed.



Finishing Touches: Squared Inputs

It may be a good idea to square the joystick inputs, as it may provide better control.

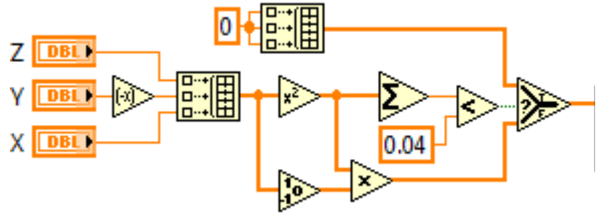
After squaring the inputs, it is necessary to restore the signs, otherwise all inputs will be positive.



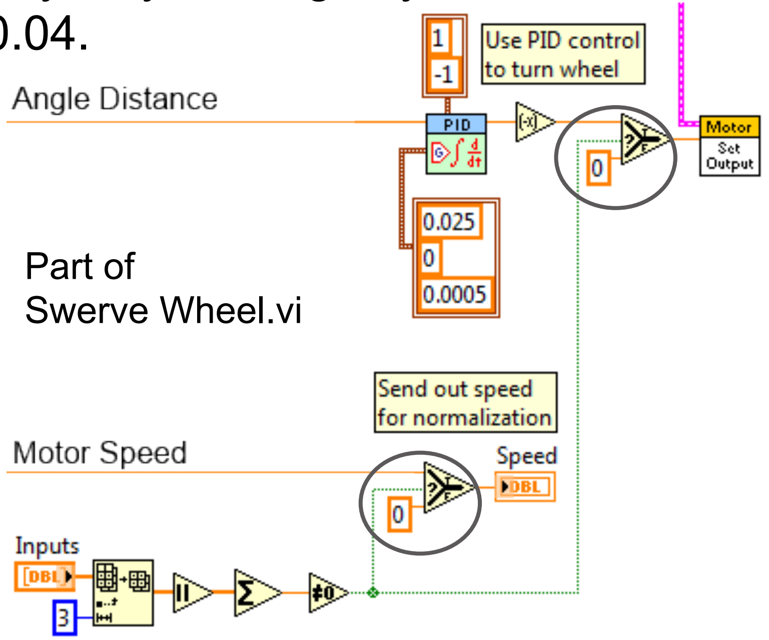
Finishing Touches: Deadband

The deadband improves control by only processing inputs over a certain magnitude. In Swerve v3, deadband is done by only moving any motors if the sum of the squared inputs are greater than 0.04.

Part of Swerve Drive.vi

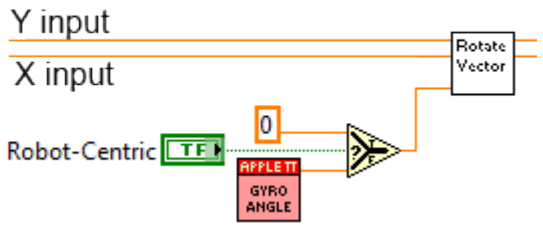


Part of Swerve Wheel.vi

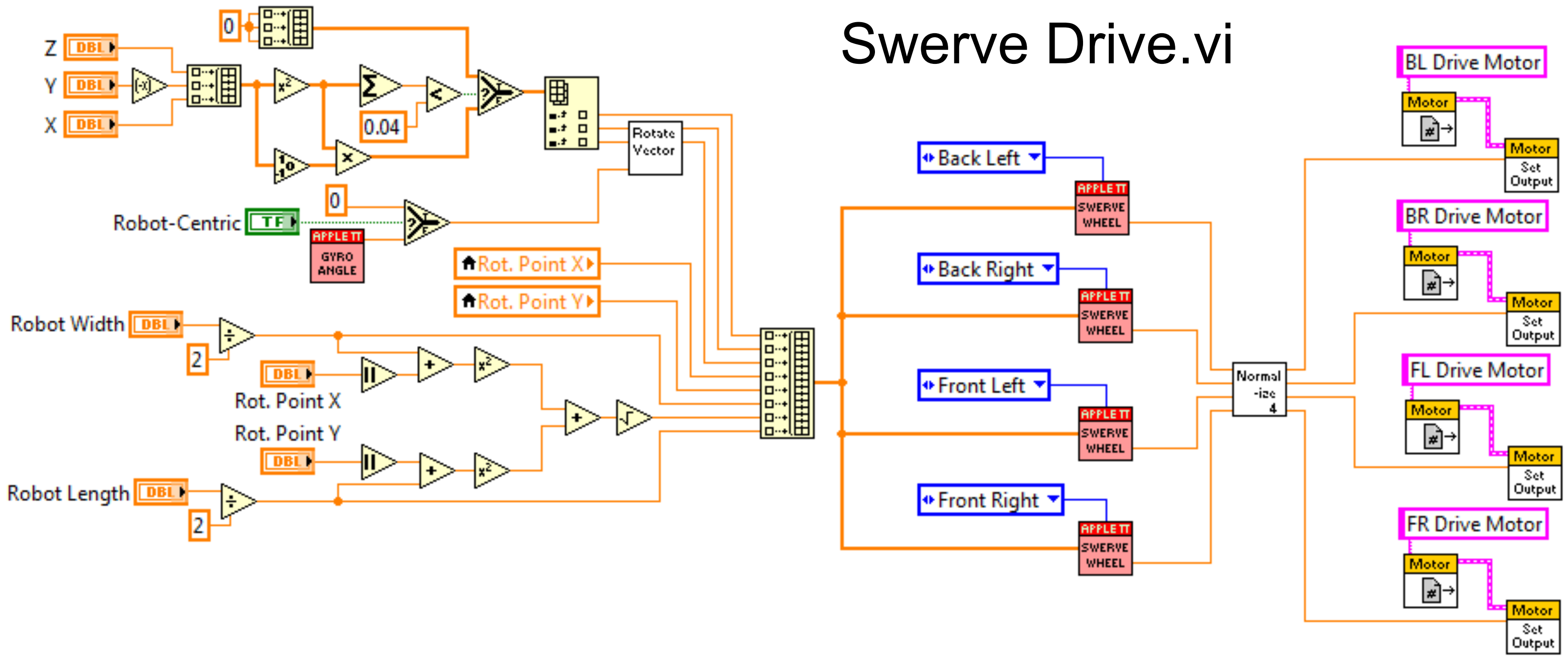


Finishing Touches: Gyro Toggle

It is good to have a feature to disable the gyro and drive in robot-centric mode. This could be useful if the gyro malfunctions, or even if the driver wants to drive in a robot-centric mode. Adding this is very simple.

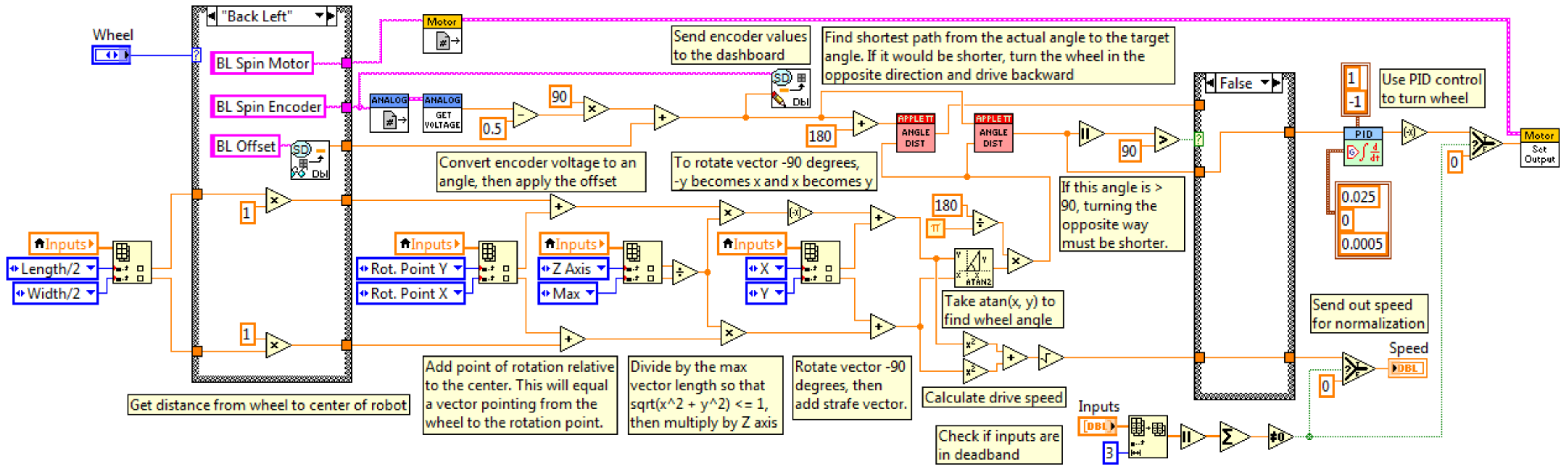


The Finished Product



The Finished Product

Swerve Wheel.vi



Send encoder values to the dashboard

Find shortest path from the actual angle to the target angle. If it would be shorter, turn the wheel in the opposite direction and drive backward

Use PID control to turn wheel

Convert encoder voltage to an angle, then apply the offset

To rotate vector -90 degrees, -y becomes x and x becomes y

If this angle is > 90, turning the opposite way must be shorter.

Take atan(x, y) to find wheel angle

Send out speed for normalization

Get distance from wheel to center of robot

Add point of rotation relative to the center. This will equal a vector pointing from the wheel to the rotation point.

Divide by the max vector length so that sqrt(x^2 + y^2) <= 1, then multiply by Z axis

Rotate vector -90 degrees, then add strafe vector.

Calculate drive speed

Check if inputs are in deadband